Cloud Security Handbook

Find out how to effectively secure cloud environments using AWS, Azure, and GCP

Eyal Estrin

Cloud Security Handbook

Find out how to effectively secure cloud environments using AWS, Azure, and GCP

Eyal Estrin

Packt

BIRMINGHAM—MUMBAI

Cloud Security Handbook

Group Product Manager: Rahul Nair

Publishing Product Manager: Rahul Nair

Senior Editor: Arun Nadar

Content Development Editor: Sulagna Mohanty

Technical Editor: Arjun Varma

Copy Editor: Safis Editing

Project Coordinator: Shagun Saini

Proofreader: Safis Editing

Indexer: Pratik Shirodkar

Production Designer: Joshua Misquitta

Marketing Coordinator: Hemangi Lotlikar

First published: March 2022

Production reference: 1100322

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-80056-919-5

www.packt.com

I wish to dedicate this book to my loving wife for all the support she provided me with during the long hours spent writing this book.

– Eyal Estrin

Contributors

About the author

Eyal Estrin is a cloud security architect who has been working with cloud services since 2015. He has been involved in the design and implementation of cloud environments from both the IT and security aspects.

He has worked with AWS, Azure, and Google Cloud in a number of different organizations (in the banking, academia, and healthcare sectors).

He has attained several top cloud security certifications – CCSP, CCSK, and AWS.

He shares his knowledge through social media (LinkedIn, Twitter, Medium, and more) for the benefit of cloud experts around the world.

About the reviewers

Randy M. Black is a 25-year veteran in the IT industry and an early adopter of DevOps. Randy has spent the last decade working in some form or other of cloud technology and security. He abhors the silos that traditional IT creates and the detriment they pose to organizations. Randy is a strong advocate of transferring knowledge without fear of being transparent, misunderstood, or seemingly odd.

To Jesus, my rock and salvation, for His grace and peace in accompanying me through this crazy, upside-down world. And to my wife, Jill, who has supported and stood by me in everything that I do, and who is the cornerstone of my success. And finally, to my four children, who don't always understand what I do, but appreciate the fact that I am doing it.

Timothy Orr (`@easttim0r` on Twitter) designs, builds, and operates secure systems in complex cloud environments. He supports customers with cloud security automation, serverless architecture, threat detection and response, security analysis, and multi-tenant cloud brokering and governance. Tim holds a master's degree in InfoSec, CISSP, AWS Security Specialty, AWS Solutions Architect Professional, and AWS SysOps Administrator Associate certifications.

Table of Contents

3
Securing Storage Services

4
Securing Networking Services

Section 2: Deep Dive into IAM, Auditing, and Encryption

5

Effective Strategies to Implement IAM Solutions

6

Monitoring and Auditing Your Cloud Environments

7

Applying Encryption in Cloud Services

Section 3: Threats and Compliance Management

8

Understanding Common Security Threats to Cloud Services

9

Handling Compliance and Regulation

10
Engaging with Cloud Providers

Section 4: Advanced Use of Cloud Services

11
Managing Hybrid Clouds

12

Managing Multi-Cloud Environments

13

Security in Large-Scale Environments

Preface

Cloud Security Handbook provides complete coverage of security aspects when designing, building, and maintaining environments in the cloud. This book is filled with best practices to help you smoothly transition to the public cloud, while keeping your environments secure. You do not have to read everything - simply find out which cloud provider is common at your workplace, or which cloud provider you wish to focus on, and feel free to skip the rest.

Who this book is for

This book is for IT or information security personnel taking their first steps in the public cloud or migrating existing environments to the cloud. DevOps professionals, cloud engineers, or cloud architects maintaining production environments in the cloud will also benefit from this book.

What this book covers

Chapter 1, *Introduction to Cloud Security*, in order to give you a solid understanding of cloud security, helps you to understand concepts such as **Infrastructure as a Service** (**IaaS**), **Platform as a Service** (**PaaS**), **Software as a Service** (**SaaS**), private cloud, public cloud, hybrid cloud, multi-cloud, and the Shared Responsibility Model. This and the rest of the chapters in this book will allow you to understand how to implement security in various cloud environments.

Chapter 2, *Securing Compute Services*, covers how **Amazon Web Services** (**AWS**), Microsoft Azure, and **Google Cloud Platform** (**GCP**) implement virtual machines, managed databases, containers, Kubernetes, and serverless architectures, and what the best practices for securing those services are.

Chapter 3, *Securing Storage Services*, covers how AWS, Microsoft Azure, and GCP implement object storage, block storage, and managed file storage, and what the best practices for securing those services are.

Chapter 4, *Securing Network Services*, covers how AWS, Microsoft Azure, and GCP implement virtual networks, security groups, DNS services, CDN, VPN services, DDoS protection services, and web application firewalls, and what the best practices for securing those services are.

Chapter 5, *Effective Strategies to Implement IAM Solutions*, covers how AWS, Microsoft Azure, and GCP implement directory services, how these cloud providers implement identity and access management for modern cloud applications, how to implement multi-factor authentication, and how to secure all these services.

Chapter 6, *Monitoring and Auditing of Your Cloud Environment*, covers how AWS, Microsoft Azure, and GCP implement audit mechanisms, how to detect threats in automated and large-scale environments, and how to capture network traffic for troubleshooting and security incident detection (digital forensics).

Chapter 7, *Applying Encryption in Cloud Services*, covers when to use symmetric and asymmetric encryption in a cloud environment, what the various alternatives for key management services in AWS, Azure, and GCP are, what the alternatives and best practices for storing secrets in code are, and how to implement encryption in traffic and encryption at rest on the AWS, Azure, and GCP cloud services.

Chapter 8, *Understanding Common Security Threats to Cloud Computing*, covers what the common security threats in public cloud environments are, how to detect those threats, and what the countermeasures to mitigate such threats using built-in services in AWS, Azure, and GCP are.

Chapter 9, *Handling Compliance and Regulation*, covers what the common security standards related to cloud environments are, what the different levels of **Security Operations Center** (**SOC**) are, and how to use cloud services to comply with the European data privacy regulation, GDPR.

Chapter 10, *Engaging with Cloud Providers*, covers how to conduct a risk assessment in a public cloud environment, what the important questions to ask a cloud provider prior to the engagement phase are, and what important topics to embed inside a contractual agreement with the cloud provider.

Chapter 11, *Managing Hybrid Clouds*, covers how to implement common features such as identity and access management, patch management, vulnerability management, configuration management, monitoring, and network security aspects in hybrid cloud environments.

Chapter 12, *Managing Multi-Cloud Environments*, covers how to implement common topics such as identity and access management, patch management, vulnerability management, configuration management, monitoring, and network security aspects in multi-cloud environments.

Chapter 13, *Security in Large-Scale Environments*, covers what the common **Infrastructure as a Code** (**IaC**) alternatives are, how to implement patch management in a centralized manner, how to control configuration and compliance management, and how to detect vulnerabilities in cloud environments (managed services and sample tools) in a large production environment.

To get the most out of this book

The following are some of the requirements to get the most out of the book:

Software/hardware covered in the book	OS requirements
An up-to-date web browser	Windows, macOS, or Linux (any)
Credentials to access the AWS, Azure, or GCP web console	

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://static.packt-cdn.com/downloads/9781800569195_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "If a resource node has set `inheritFromParent = true`, then the effective policy of the parent resource is inherited."

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "**Azure Event Hubs**: This is for sending audit logs to an external SIEM system for further analysis."

> **Tips or Important Notes**
> Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the Internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share your thoughts

Once you've read *Cloud Security Handbook*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Section 1: Securing Infrastructure Cloud Services

On completion of this part, you will have a solid understanding of how to secure the basic building blocks of cloud services (cloud deployment and service models, compute, storage, and network)

This part of the book comprises the following chapters:

- *Chapter 1*, *Introduction to Cloud Security*
- *Chapter 2*, *Securing Compute Services*
- *Chapter 3*, *Securing Storage Services*
- *Chapter 4*, *Securing Network Services*

1
Introduction to Cloud Security

This book, *Cloud Security Techniques and Best Practices*, is meant for various audiences. You could be taking your first steps working with cloud services, or you could be coming from an IT perspective and want to know about various compute and storage services and how to configure them securely. Or, you might be working in information security and want to know the various authentication, encryption, and audit services and how to configure them securely, or you might be working with architecture and want to know how to design large-scale environments in the cloud in a secure way.

Reading this book will allow you to make the most of cloud services while focusing on security aspects. Before discussing cloud services in more detail, let me share my opinion regarding cloud services.

The world of IT is changing. For decades, organizations used to purchase physical hardware, install operating systems, and deploy software. This routine required a lot of ongoing maintenance (for patch deployment, backup, monitoring, and so on).

The cloud introduced a new paradigm – that is, the ability to consume managed services to achieve the same goal of running software (from file servers to **Enterprise Resource Planning** (**ERP)** or **Customer Relationship Management** (**CRM**) products), while using the expertise of the hyper-scale cloud providers.

Some well-known use cases of cloud computing are as follows:

- **Netflix** – one of the largest video streaming services world-wide. It uses AWS to run its media streaming services:

  `https://aws.amazon.com/solutions/case-studies/netflix-case-study`

- **Mercedes-Benz** – one of the most famous automotive brands. It uses Azure to run its research and development:

  `https://customers.microsoft.com/en-us/story/784791-mercedes-benz-r-and-d-creates-container-driven-cars-powered-by-microsoft-azure`

- **Home Depot** – the largest home improvement retailer in the United States. It uses Google Cloud to run its online stores:

  `https://cloud.google.com/customers/featured/the-home-depot`

In this book, we will compare various aspects of cloud computing (from fundamental services such as compute, storage, and networking, to compliance management and best practices for building and maintaining large-scale environments in a secure way), while reviewing the different alternatives offered by **Amazon Web Services** (**AWS**), **Microsoft Azure**, and **Google Cloud Platform** (**GCP**).

It does not matter which organization you are coming from – this book will allow you to have a better understanding of how to achieve security in any of the large hyper-scale cloud providers.

You do not have to read everything – simply find out which cloud provider is common at your workplace or which cloud provider you wish to focus on, and feel free to skip the rest.

In this chapter, we will cover the following topics:

- Why we need security
- Cloud service models
- Cloud deployment models
- The shared responsibility model

Technical requirements

This chapter is an introduction to cloud security, so there are no technical requirements.

What is a cloud service?

As part of this introduction, let's define the terminology to make sure we are all on the same page.

The **National Institute of Standards and Technology** (**NIST**) defines *cloud* as a technology that has the following five characteristics:

- **On-demand self-service**: Imagine you wish to open a blog and you need compute resources. Instead of purchasing hardware and waiting for the vendor to ship it to your office and having to deploy software, the easier alternative can be a self-service portal, where you can select a pre-installed operating system and content management system that you can deploy within a few minutes by yourself.
- **Broad network access**: Consider having enough network access (the type that large **Internet Service Providers** (**ISPs**) have) to serve millions of end users with your application.
- **Resource pooling**: Consider having thousands of computers, running in a large server farm, and being able to maximize their use (from CPU, memory, and storage capacity), instead of having a single server running 10% of its CPU utilization.
- **Rapid elasticity**: Consider having the ability to increase and decrease the amount of compute resources (from a single server to thousands of servers, and then back to a single server), all according to your application or service needs.
- **Measured service**: Consider having the ability to pay for only the resources you consumed and being able to generate a billing report that shows which resources have been used and how much you must pay for the resources.

Further details relating to the NIST definition can be found at the following link:

`https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf`

What are the cloud deployment models?

Now that we understand what the cloud characteristics are, let's talk about cloud deployment models:

- **Private cloud**: An infrastructure deployed and maintained by a single organization. Let's say we are a large financial organization (such as a bank or insurance organization), we would like to serve various departments in our organization (from HR, IT, sales, and so on), and we might have regulatory requirements to keep customers' data on-premises – a private cloud can be a suitable solution.

- **Public cloud**: An infrastructure deployed and maintained by a service provider for serving multiple customers and organizations, mostly accessible over the internet. Naturally, this book will focus on the public cloud model, with reference to various services offered by AWS, Azure, and GCP.
- **Hybrid cloud**: A combination of a private cloud (or on-premises cloud) and at least one public cloud infrastructure. I like to consider the hybrid cloud as an extension of the local data center. We should not consider this extension as something separate, and we should protect it the same way we protect our local data center.
- **Multi-cloud**: A scenario where our organization is either using multiple managed services (see the definition of **SaaS** in the next section) or using multiple public cloud infrastructure (see the definitions of **IaaS** and **PaaS** in the next section).

What are the cloud service models?

An essential part of understanding clouds is understanding the three cloud service models:

- **Infrastructure as a Service** (**IaaS**): This is the most fundamental service model, where a customer can select the virtual machine size (in terms of the amount of CPU and memory), select a pre-configured operating system, and deploy software inside the virtual machine instance according to business needs (services such as **Amazon EC2**, **Azure Virtual Machines**, and **Google Compute Engine**).
- **Platform as a Service** (**PaaS**): This type of service model varies from managed database services to managed application services (where a customer can import code and run it inside a managed environment) and more (services such as **AWS Elastic Beanstalk**, **Azure Web Apps**, and **Google App Engine**).
- **Software as a Service** (**SaaS**): This is the most widely used service model – a fully managed software environment where, as a customer, you usually open a web browser, log in to an application, and consume services. These could be messaging services, ERP, CRM, business analytics, and more (services such as **Microsoft Office 365**, **Google Workspaces**, **Salesforce CRM**, **SAP SuccessFactors**, and **Oracle Cloud HCM**).

Understanding the cloud service models will allow you to understand your role as a customer, explained later in the *What is the shared responsibility model?* section.

Why we need security

As mentioned previously, we can see clear benefits of using cloud services that enable our business to focus on what brings us value (from conducting research in a pharmaceutical lab, to selling products on a retail site, and so on).

But what about security? And, specifically, cloud security?

Why should our organization focus on the overhead called *information security* (and, in the context of this book, *cloud security*)?

The cloud has changed the paradigm of organizations controlling their data on-premises (from HR data to customers' data) and investing money in maintaining data centers, servers, storage, network equipment, and the application tier.

Using public clouds has changed the way organizations look at information security (in the context of this book, cloud security).

The following are a few common examples of the difference between on-premises data solutions and the cloud:

	Traditional data center	**Cloud environment**
Who is responsible for physical security?	The organization	The cloud service provider
Where is the data located?	On-premises data center	Cloud provider's data center
Who controls access to the data?	The organization	Both the organization and the cloud service provider
Who is responsible for vulnerability management and patch management?	The organization	IaaS – The organization PaaS – The cloud provider SaaS – The cloud provider
Who is in charge of incident response?	The organization	Both the organization and the cloud service provider
Who is responsible for law, privacy, and regulation?	The organization	Both the organization and the cloud service provider

Table 1.1 – Differences between on-premises data solutions and the cloud

Organizations are often unwilling to migrate to a public cloud for security reasons because the physical servers are located outside of the organization's direct control, and sometimes even outside their physical geography.

Here are a few questions often asked by organizations' management:

- Are my servers going to behave the same as if they were on-premises?
- How do I protect my servers outside my data center from a data breach?
- How do I know the cloud provider will not have access to my data?
- Do my employees have enough knowledge to work in new environments such as the public cloud?

Perhaps the most obvious question asked is – *is the public cloud secure enough to store my data?*

From my personal experience, the answer is *yes*.

By design, the hyper-scale cloud providers invest billions of dollars protecting their data centers, building secure services, investing in employee training, and locating security incidents and remediating them fast. This is all with much higher investment, attention, and expertise than most organizations can dedicate to protecting their local data centers.

The reason for this is simple – if a security breach happens to one of the hyper-scale cloud providers, their customers' trust will be breached, and the cloud providers will run out of business.

At the end of the day, cloud security enables our organization to achieve (among other things) the following:

- **Decreased attack surface**: Using central authentication, data encryption, DDoS protection services, and more
- **Compliance with regulation**: Deploying environments according to best practices
- **Standardization and best practices**: Enforcing security using automated tools and services

Reading this book will allow you to have a better understanding of various methods to secure your cloud environments – most of them using the cloud vendor's built-in services and capabilities.

What is the shared responsibility model?

When speaking about cloud security and cloud service models (IaaS/PaaS/SaaS), the thing that we all hear about is the **shared responsibility model**, which tries to draw a line between the cloud provider and the customer's responsibilities regarding security.

As you can see in the following diagram, the cloud provider is always responsible for the lower layers – from the physical security of their data centers, through networking, storage, host servers, and the virtualization layers:

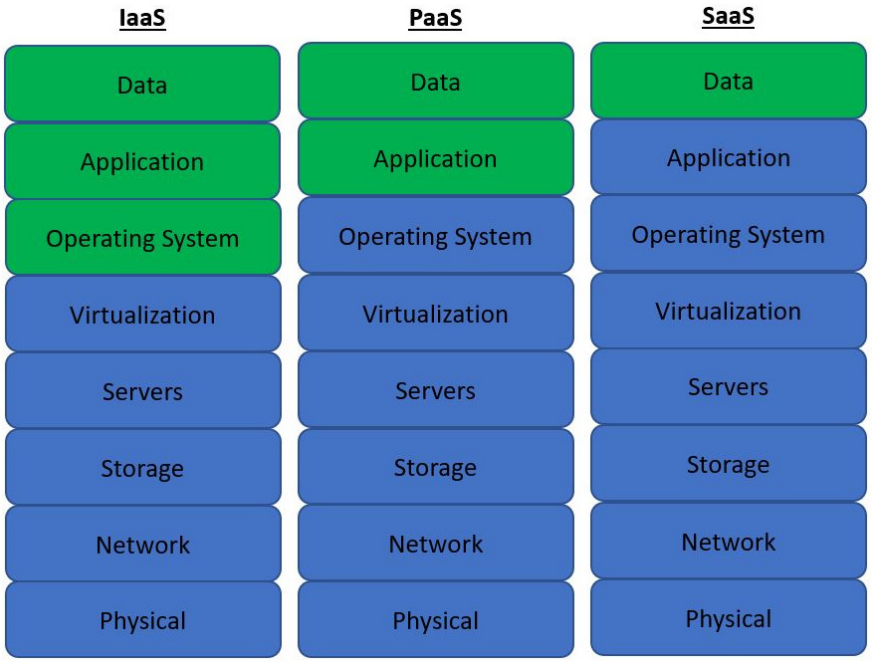

Figure 1.1 – The shared responsibility model

Above the virtualization layer is where the responsibility begins to change.

When working with IaaS, we, as the customers, can select a pre-installed image of an operating system (with or without additional software installed inside the image), deploy our applications, and manage permissions to access our data.

When working with PaaS, we, as the customers, may have the ability to control code in a managed environment (services such as AWS Elastic Beanstalk, Azure Web Apps, and Google App Engine) and manage permissions to access our data.

When working with SaaS, we, as the customers, received a fully managed service, and all we can do is manage permissions to access our data.

In the next sections, we will look at how the various cloud providers (AWS, Azure, and GCP) look at the shared responsibility model from their own perspective.

For more information on the shared responsibility model, you can check the following link: `https://tutorials4sharepoint.wordpress.com/2020/04/24/shared-responsibility-model/`.

AWS and the shared responsibility model

Looking at the shared responsibility model from AWS's point of view, we can see the clear distinction between AWS's responsibility for the security *of* the cloud (physical hardware and the lower layers such as host servers, storage, database, and network) and the customer's responsibility for security *in* the cloud (everything the customer controls – operating system, data encryption, network firewall rules, and customer data). The following diagram depicts AWS and the shared responsibility model:

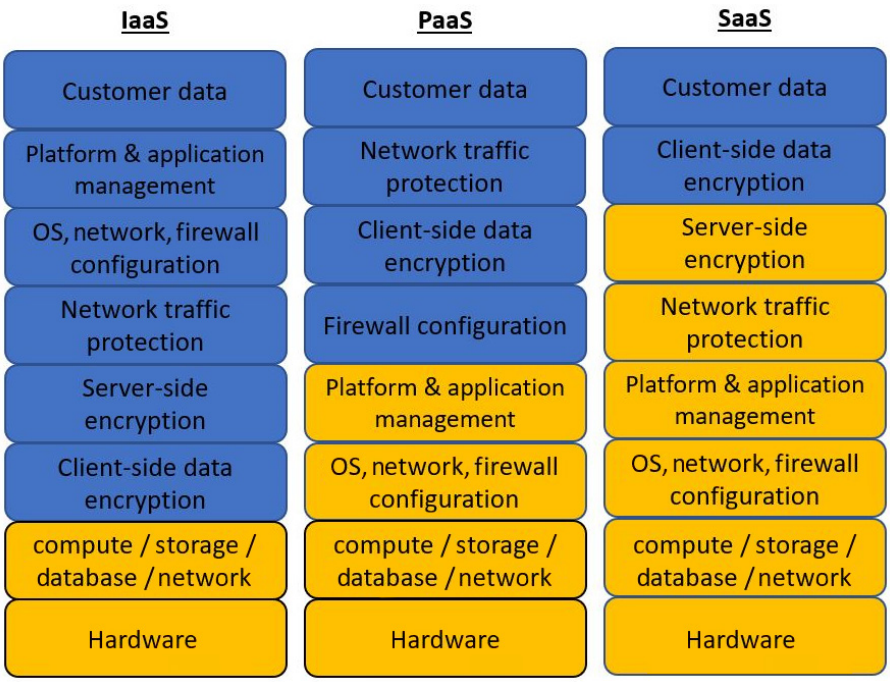

Figure 1.2 – AWS and the shared responsibility model

As a customer of AWS, reading this book will allow you to gain the essential knowledge and best practices for using common AWS services (including compute, storage, networking, authentication, and so on) in a secure way.

More information on the AWS shared responsibility model can be found at the following link: `https://aws.amazon.com/blogs/industries/applying-the-aws-shared-responsibility-model-to-your-gxp-solution/`.

Azure and the shared responsibility model

Looking at the shared responsibility model from Azure's point of view, we can see the distinction between Azure's responsibility for its data centers (physical layers) and the customer's responsibility at the top layers (identities, devices, and customers' data). In the middle layers (operating system, network controls, and applications) the responsibility changes between Azure and the customers, according to various service types. The following diagram depicts Azure and the shared responsibility model:

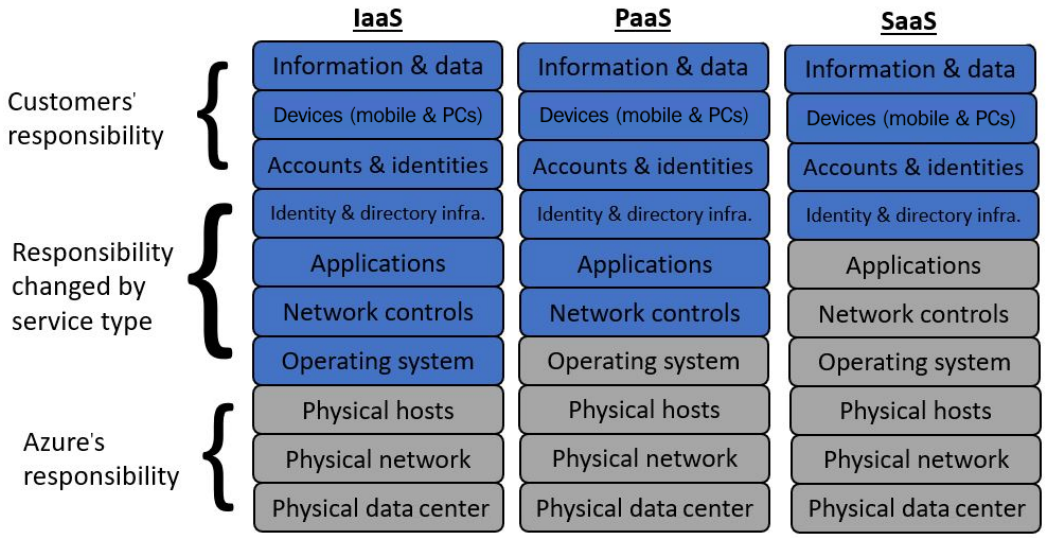

Figure 1.3 – Azure and the shared responsibility model

As a customer of Azure, reading this book will allow you to gain the essential knowledge and best practices for using common Azure services (including compute, storage, networking, authentication, and others) in a secure way.

More information on the Azure shared responsibility model can be found at the following link: `https://docs.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility`.

GCP and the shared responsibility model

Looking at the shared responsibility model from GCP's point of view, we can see that Google would like to emphasize that it builds its own hardware, which enables the company to control the hardware, boot, and kernel of its platform, including the storage layer encryption, network equipment, and logging of everything that Google is responsible for.

When looking at things that the customer is responsible for we can see a lot more layers, including everything from the guest operating system, network security rules, authentication, identity, and web application security, to things such as deployment, usage, access policies, and content (customers' data). The following diagram depicts GCP and the shared responsibility model:

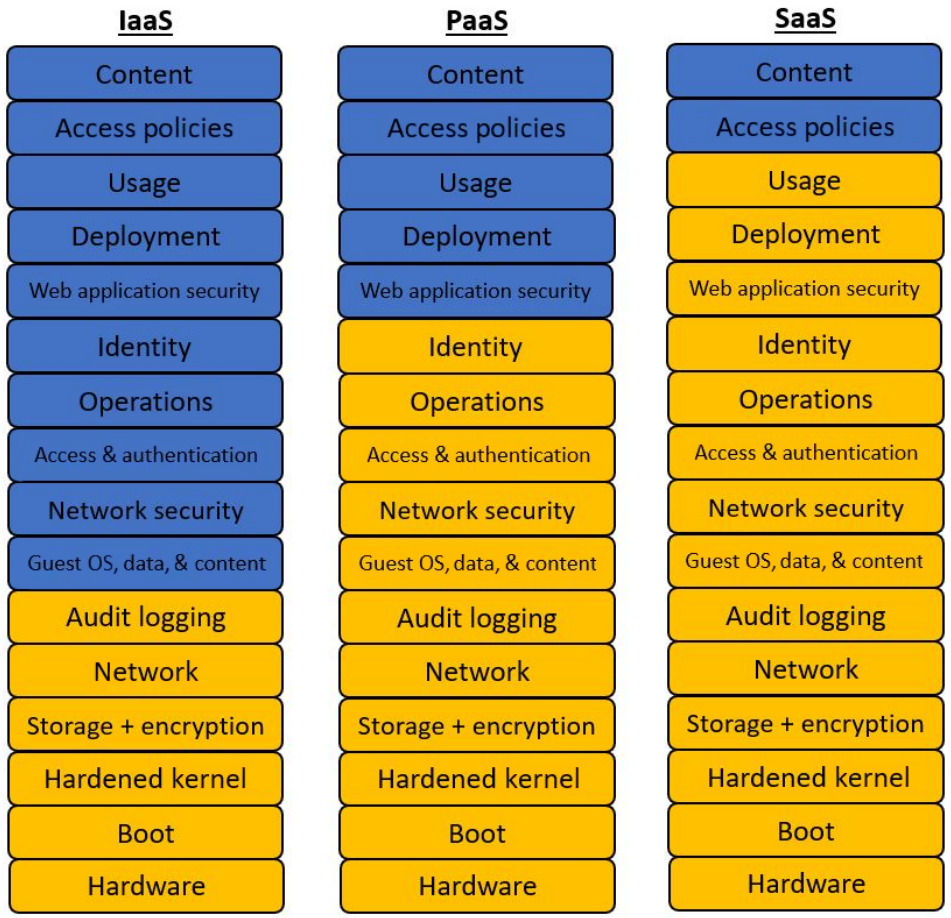

Figure 1.4 – GCP and the shared responsibility model

As a customer of GCP, reading this book will allow you to gain the essential knowledge and best practices for using common GCP services (including compute, storage, networking, authentication, and more) in a secure way.

More information about the GCP shared responsibility model can be found at the following link: `https://services.google.com/fh/files/misc/google-cloud-security-foundations-guide.pdf`.

As a customer, understanding the shared responsibility model allows you, at any given time, to understand which layers are under the cloud vendor's responsibility and which layers are under the customer's responsibility.

Command-line tools

One of the things that makes cloud environments so robust is the ability to control almost anything using the **Application Programming Interface** (**API**) or using the command line.

Most mature cloud providers have already published and maintain their own **Command-Line Interface** (**CLI**) to allow customers to perform actions in an easy and standard way.

An alternative to using the command line to interact with the cloud provider's API is using a **Software Developer Kit** (**SDK**) – a method to control actions (from deploying a virtual machine to encrypting storage), query information from a service (checking whether auditing is enabled for my customers logging into my web application), and more.

Since this book doesn't require previous development experience, I will provide examples for performing actions using the command-line tools.

During various chapters of this book, I will provide you with examples of commands that will allow you to easily implement the various security controls over AWS, Azure, and GCP.

I highly recommend that you become familiar with those tools.

AWS CLI

AWS CLI can be installed on **Windows** (64 bit), **Linux** (both x86 and ARM processors), **macOS**, and even inside a **Docker** container.

The AWS CLI documentation explains how to install the tool and provides a detailed explanation of how to use it.

The documentation can be found at `https://aws.amazon.com/cli`.

Azure CLI

Azure CLI can be installed on Windows, Linux (**Ubuntu**, **Debian**, **RHEL**, **CentOS**, **Fedora**, **openSUSE**), and macOS.

The Azure CLI documentation explains how to install the tool and provides a detailed explanation of how to use it.

The documentation can be found at `https://docs.microsoft.com/en-us/cli/azure`.

Google Cloud SDK

The Google command-line tool (**gcloud CLI**) can be installed on Windows, Linux (Ubuntu, Debian, RHEL, CentOS, Fedora), and macOS.

The Google Cloud SDK documentation explains how to install the tool and provides a detailed explanation of how to use it.

The documentation can be found at `https://cloud.google.com/sdk`.

Summary

In the first chapter of this book, we learned the definition of a *cloud*, the different cloud deployment models, and the different cloud service models.

We also learned what the shared cloud responsibility model is, and how AWS, Azure, and GCP look at this concept from their own point of view.

Lastly, we had a short introduction to the AWS, Azure, and GCP built-in command-line tools, and, during the next chapters, I will provide you with examples of how to implement various tasks using the command-line tools.

This introduction will be referred to in the following chapters, where we will dive deeper into the best practices for securing cloud services using (in most cases) the cloud providers' built-in capabilities.

Securing cloud environments can be challenging, depending on your previous knowledge in IT or information security or cloud services in general.

Reading this book will assist you in gaining the necessary knowledge of how to secure cloud environments, regardless of your role in the organization or your previous experience.

In the next chapter, we will review the various compute services in the cloud (including virtual machines, managed databases, container services, and finally serverless services).

2
Securing Compute Services

Speaking about cloud services, specifically **Infrastructure as a Service** (**IaaS**), the most common resource everyone talks about is compute – from the traditional **virtual machines** (**VMs**), through managed databases (run on VMs on the backend), to modern compute architecture such as containers and eventually serverless.

This chapter will cover all types of compute services and provide you with best practices on how to securely deploy and manage each of them.

In this chapter, we will cover the following topics:

- Securing VMs (authentication, network access control, metadata, serial console access, patch management, and backups)
- Securing Managed Database Services (identity management, network access control, data protection, and auditing and monitoring)
- Securing Containers (identity management, network access control, auditing and monitoring, and compliance)
- Securing serverless/function as a service (identity management, network access control, auditing and monitoring, compliance, and configuration change)

Technical requirements

For this chapter, you need to have an understanding of VMs, what managed databases are, and what containers (and Kubernetes) are, as well as a fundamental understanding of serverless.

Securing VMs

Each cloud provider has its own implementation of VMs (or virtual servers), but at the end of the day, the basic idea is the same:

1. Select a machine type (or size) – a ratio between the amount of **virtual CPU** (**vCPU**) and memory, according to their requirements (general-purpose, compute-optimized, memory-optimized, and so on).
2. Select a preinstalled image of an operating system (from Windows to Linux flavors).
3. Configure storage (adding additional volumes, connecting to file sharing services, and others).
4. Configure network settings (from network access controls to micro-segmentation, and others).
5. Configure permissions to access cloud resources.
6. Deploy an application.
7. Begin using the service.
8. Carry out ongoing maintenance of the operating system.

According to the shared responsibility model, when using IaaS, we (as the customers) are responsible for the deployment and maintenance of virtual servers, as explained in the coming section.

Next, we are going to see what the best practices are for securing common VM services in AWS, Azure, and GCP.

Securing Amazon Elastic Compute Cloud (EC2)

Amazon EC2 is the Amazon VM service.

General best practices for EC2 instances

Following are some of the best practices to keep in mind:

- Use only trusted AMI when deploying EC2 instances.

- Use a minimal number of packages inside an AMI, to lower the attack surface.
- Use Amazon built-in agents for EC2 instances (backup, patch management, hardening, monitoring, and others).
- Use the new generation of EC2 instances, based on the AWS Nitro System, which offloads virtualization functions (such as network, storage, and security) to dedicated software and hardware chips. This allows the customer to get much better performance, with much better security and isolation of customers' data.

For more information, please refer the following resources:

Best practices for building AMIs:

`https://docs.aws.amazon.com/marketplace/latest/userguide/best-practices-for-building-your-amis.html`

Amazon Linux AMI:

`https://aws.amazon.com/amazon-linux-2/`

AWS Nitro System:

`https://aws.amazon.com/ec2/nitro/`

Best practices for authenticating to an instance

AWS does not have access to customers' VMs.

It doesn't matter whether you choose to deploy a Windows or a Linux machine, by running the EC2 launch deployment wizard, you must choose either an existing key pair or create a new key. This set of private/public keys is generated at the client browser – AWS does not have any access to these keys, and therefore cannot log in to your EC2 instance.

For Linux instances, the key pair is used for logging in to the machine via the SSH protocol.

Refer to the following link: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html`.

For Windows instances, the key pair is used to retrieve the built-in administrator's password.

Refer to the following link: `https://docs.amazonaws.cn/en_us/AWSEC2/latest/WindowsGuide/ec2-windows-passwords.html`.

The best practices are as follows:

- Keep your private keys in a secured location. A good alternative for storing and retrieving SSH keys is to use AWS Secrets Manager.
- Avoid storing private keys on a bastion host or any instance directly exposed to the internet. A good alternative to logging in using SSH, without an SSH key, is to use AWS Systems Manager, through Session Manager.
- Join Windows or Linux instances to an **Active Directory** (**AD**) domain and use your AD credentials to log in to the EC2 instances (and avoid using local credentials or SSH keys completely).

For more information, please refer the following resources:

How to use AWS Secrets Manager to securely store and rotate SSH key pairs:

`https://aws.amazon.com/blogs/security/how-to-use-aws-secrets-manager-securely-store-rotate-ssh-key-pairs/`

Allow SSH connections through Session Manager:

`https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager-getting-started-enable-ssh-connections.html`

Seamlessly join a Windows EC2 instance:

`https://docs.aws.amazon.com/directoryservice/latest/admin-guide/launching_instance.html`

Seamlessly join a Linux EC2 instance to your AWS-managed Microsoft AD directory:

`https://docs.aws.amazon.com/directoryservice/latest/admin-guide/seamlessly_join_linux_instance.html`

Best practices for securing network access to an instance

Access to AWS resources and services such as EC2 instances is controlled via *security groups* (at the EC2 instance level) or a **network access control list** (**NACL**) (at the subnet level), which are equivalent to the on-premises layer 4 network firewall or access control mechanism.

As a customer, you configure parameters such as source IP (or CIDR), destination IP (or CIDR), destination port (or predefined protocol), and whether the port is TCP or UDP.

You may also use another security group as either the source or destination in a security group.

For remote access and management of Linux machines, limit inbound network access to TCP port `22`.

For remote access and management of Windows machines, limit inbound network access to TCP port `3389`.

The best practices are as follows:

- For remote access protocols (SSH/RDP), limit the source IP (or CIDR) to well-known addresses. Good alternatives for allowing remote access protocols to an EC2 instance are to use a VPN tunnel, use a bastion host, or use AWS Systems Manager Session Manager.
- For file sharing protocols (CIFS/SMB/FTP), limit the source IP (or CIDR) to well-known addresses.
- Set names and descriptions for security groups to allow a better understanding of the security group's purpose.
- Use tagging (that is, *labeling*) for security groups to allow a better understanding of which security group belongs to which AWS resources.
- Limit the number of ports allowed in a security group to the minimum required ports for allowing your service or application to function.

For more information, please refer the following resources:

Amazon EC2 security groups for Linux instances: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-security-groups.html`

Security groups for your **virtual private cloud** (**VPC**): `https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html`

AWS Systems Manager Session Manager:

`https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html`

Compare security groups and network ACLs:

`https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html#VPC_Security_Comparison`

Best practices for securing instance metadata

Instance metadata is a method to retrieve information about a running instance, such as the hostname and internal IP address.

An example of metadata about a running instance can be retrieved from within an instance, by either opening a browser from within the operating system or using the command line, to a URL such as `http://169.254.169.254/latest/meta-data/`.

Even though the IP address is an internal IP (meaning it cannot be accessed from outside the instance), the information, by default, can be retrieved locally without authentication.

AWS allows you to enforce authenticated or session-oriented requests to the instance metadata, also known as **Instance Metadata Service Version 2** (**IMDSv2**).

The following command uses the AWS CLI tool to enforce IMDSv2 on an existing instance:

```
aws ec2 modify-instance-metadata-options \
    --instance-id <INSTANCE-ID> \
    --http-endpoint enabled --http-tokens required
```

For more information, please refer the following resource:

Configure the instance metadata service:

`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html`

Best practices for securing a serial console connection

For troubleshooting purposes, AWS allows you to connect using a *serial console* (a similar concept to what we used to have in the physical world with network equipment) to resolve network or operating system problems when SSH or RDP connections are not available.

The following command uses the AWS CLI tool to allow serial access at the AWS account level to a specific AWS Region:

```
aws ec2 enable-serial-console-access --region <Region_Code>
```

Since this type of remote connectivity exposes your EC2 instance, it is recommended to follow the following best practices:

- Access to the EC2 serial console should be limited to the group of individuals using **identity and access management** (**IAM**) roles.
- Only allow access to EC2 serial console when required.
- Always set a user password on an instance before allowing the EC2 serial console.

For more information, please refer the following resource:

Configure access to the EC2 serial console:

`https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configure-access-to-serial-console.html`

Best practices for conducting patch management

Patch management is a crucial part of every instance of ongoing maintenance.

To deploy security patches for either Windows or Linux-based instances in a standard manner, it is recommended to use AWS Systems Manager Patch Manager, following this method:

1. Configure the patch baseline.
2. Scan your EC2 instances for deviation from the patch baseline at a scheduled interval.
3. Install missing security patches on your EC2 instances.
4. Review the Patch Manager reports.

The best practices are as follows:

- Use AWS Systems Manager Compliance to make sure all your EC2 instances are up to date.
- Create a group with minimal IAM privileges to allow only relevant team members to conduct patch deployment.
- Use tagging (that is, *labeling*) for your EC2 instances to allow patch deployment groups per tag (for example, *prod* versus *dev* environments).
- For stateless EC2 instances (where no user session data is stored inside an EC2 instance), replace an existing EC2 instance with a new instance, created from an up-to-date operating system image.

For more information, please refer the following resource:

Software patching with AWS Systems Manager:

`https://aws.amazon.com/blogs/mt/software-patching-with-aws-systems-manager/`

Best practices for securing backups

Backing up is crucial for EC2 instance recovery.

The AWS Backup service encrypts your backups in transit and at rest using AWS encryption keys, stored in AWS **Key Management Service** (**KMS**) (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*), as an extra layer of security, independent of your **Elastic Block Store** (**EBS**) volume or snapshot encryption keys.

The best practices are as follows:

- Configure the AWS Backup service with an IAM role to allow access to the encryption keys stored inside AWS KMS.
- Configure the AWS Backup service with an IAM role to allow access to your backup vault.
- Use tagging (that is, *labeling*) for backups to allow a better understanding of which backup belongs to which EC2 instance.
- Consider replicating your backups to another region.

For more information, please refer the following resources:

Protecting your data with AWS Backup:

`https://aws.amazon.com/blogs/storage/protecting-your-data-with-aws-backup/`

Creating backup copies across AWS Regions:

`https://docs.aws.amazon.com/aws-backup/latest/devguide/cross-region-backup.html`

Summary

In this section, we have learned how to securely maintain a VM, based on AWS infrastructure – from logging in to securing network access, troubleshooting using a serial console, patch management, and backup.

Securing Azure Virtual Machines

Azure Virtual Machines is the Azure VM service.

General best practices for Azure Virtual Machines

Following are some of the best practices to keep in mind:

- Use only trusted images when deploying Azure Virtual Machines.
- Use a minimal number of packages inside an image, to lower the attack surface.
- Use Azure built-in agents for Azure Virtual Machines (backup, patch management, hardening, monitoring, and others).
- For highly sensitive environments, use Azure confidential computing images, to ensure security and isolation of customers' data.

For more information, please refer the following resources:

Azure Image Builder overview:

`https://docs.microsoft.com/en-us/azure/virtual-machines/image-builder-overview`

Using Azure for cloud-based confidential computing:

`https://docs.microsoft.com/en-us/azure/confidential-computing/overview#using-azure-for-cloud-based-confidential-computing-`

Best practices for authenticating to a VM

Microsoft does not have access to customers' VMs.

It doesn't matter whether you choose to deploy a Windows or a Linux machine, by running the **create a virtual machine** wizard, to deploy a new Linux machine, by default, you must choose either an existing key pair or create a new key pair.

This set of private/public keys is generated at the client side – Azure does not have any access to these keys, and therefore cannot log in to your Linux VM.

For Linux instances, the key pair is used for logging in to the machine via the SSH protocol.

For more information, please refer the following resource:

Generate and store SSH keys in the Azure portal:

`https://docs.microsoft.com/en-us/azure/virtual-machines/ssh-keys-portal`

For Windows machines, when running the **create a new virtual machine** wizard, you are asked to specify your own administrator account and password to log in to the machine via the RDP protocol.

For more information, please refer the following resource:

Create a Windows VM:

`https://docs.microsoft.com/en-us/azure/virtual-machines/windows/quick-create-portal#create-virtual-machine`

The best practices are as follows:

- Keep your credentials in a secured location.
- Avoid storing private keys on a bastion host (VMs directly exposed to the internet).
- Join Windows or Linux instances to an AD domain and use your AD credentials to log in to the VMs (and avoid using local credentials or SSH keys completely).

For more information, please refer the following resources:

Azure Bastion:

`https://azure.microsoft.com/en-us/services/azure-bastion`

Join a Windows Server VM to an Azure AD Domain Services-managed domain using a Resource Manager template:

`https://docs.microsoft.com/en-us/azure/active-directory-domain-services/join-windows-vm-template`

Join a Red Hat Enterprise Linux VM to an Azure AD Domain Services-managed domain:

`https://docs.microsoft.com/en-us/azure/active-directory-domain-services/join-rhel-linux-vm`

Best practices for securing network access to a VM

Access to Azure resources and services such as VMs is controlled via **network security groups**, which are equivalent to the on-premises layer 4 network firewall or access control mechanism.

As a customer, you configure parameters such as source IP (or CIDR), destination IP (or CIDR), source port (or a predefined protocol), destination port (or a predefined protocol), whether the port is TCP or UDP, and the action to take (either allow or deny).

For remote access and management of Linux machines, limit inbound network access to TCP port `22`.

For remote access and management of Windows machines, limit inbound network access to TCP port `3389`.

The best practices are as follows:

- For remote access protocols (SSH/RDP), limit the source IP (or CIDR) to well-known addresses. Good alternatives for allowing remote access protocols to an Azure VM is to use a VPN tunnel, use Azure Bastion, or use Azure **Privileged Identity Management** (**PIM**) to allow just-in-time access to a remote VM.
- For file sharing protocols (CIFS/SMB/FTP), limit the source IP (or CIDR) to well-known addresses.
- Set names for network security groups to allow a better understanding of the security group's purpose.
- Use tagging (that is, *labeling*) for network security groups to allow a better understanding of which network security group belongs to which Azure resources.
- Limit the number of ports allowed in a network security group to the minimum required ports for allowing your service or application to function.

For more information, please refer the following resources:

Network security groups:

`https://docs.microsoft.com/en-us/azure/virtual-network/network-security-groups-overview`

How to open ports to a VM with the Azure portal:

`https://docs.microsoft.com/en-us/azure/virtual-machines/windows/nsg-quickstart-portal`

Azure Bastion:

`https://azure.microsoft.com/en-us/services/azure-bastion`

What is Azure AD PIM?

`https://docs.microsoft.com/en-us/azure/active-directory/privileged-identity-management/pim-configure`

Best practices for securing a serial console connection

For troubleshooting purposes, Azure allows you to connect using a *serial console* (a similar concept to what we used to have in the physical world with network equipment) to resolve network or operating system problems when SSH or RDP connections are not available.

The following commands use the Azure CLI tool to allow serial access for the entire Azure subscription level:

```
subscriptionId=$(az account show --output=json | jq -r .id)

az resource invoke-action --action enableConsole \
    --ids "/subscriptions/$subscriptionId/providers/
Microsoft.SerialConsole/consoleServices/default" --api-
version="2018-05-01"
```

Since this type of remote connectivity exposes your VMs, it is recommended to follow the following best practices:

- Access to the serial console should be limited to the group of individuals with the *Virtual Machine Contributor* role for the VM and the *Boot diagnostics* storage account.
- Always set a user password on the target VM before allowing access to the serial console.

For more information, please refer the following resources:

Azure serial console for Linux:

`https://docs.microsoft.com/en-us/troubleshoot/azure/virtual-machines/serial-console-linux`

Azure serial console for Windows:

`https://docs.microsoft.com/en-us/troubleshoot/azure/virtual-machines/serial-console-windows`

Best practices for conducting patch management

Patch management is a crucial part of every instance of ongoing maintenance.

To deploy security patches for either Windows or Linux-based instances in a standard manner, it is recommended to use **Azure Automation Update Management**, using the following method:

1. Create an automation account.
2. Enable Update Management for all Windows and Linux machines.
3. Configure the schedule settings and reboot options.
4. Install missing security patches on your VMs.
5. Review the deployment status.

The best practices are as follows:

- Use minimal privileges for the account using Update Management to deploy security patches.
- Use update classifications to define which security patches to deploy.
- When using an Azure Automation account, encrypt sensitive data (such as variable assets).
- When using an Azure Automation account, use private endpoints to disable public network access.
- Use tagging (that is, *labeling*) for your VMs to allow defining dynamic groups of VMs (for example, *prod* versus *dev* environments).
- For stateless VMs (where no user session data is stored inside an Azure VM), replace an existing Azure VM with a new instance, created from an up-to-date operating system image.

For more information, please refer the following resources:

Azure Automation Update Management:

`https://docs.microsoft.com/en-us/azure/architecture/hybrid/azure-update-mgmt`

Manage updates and patches for your VMs:

`https://docs.microsoft.com/en-us/azure/automation/update-management/manage-updates-for-vm`

Update management permissions:

`https://docs.microsoft.com/en-us/azure/automation/automation-role-based-access-control#update-management-permissions`

Best practices for securing backups

Backing up is crucial for VM recovery.

The Azure Backup service encrypts your backups in transit and at rest using Azure Key Vault (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).

The best practices are as follows:

- Use Azure **role-based access control** (**RBAC**) to configure Azure Backup to have minimal access to your backups.
- For sensitive environments, encrypt data at rest using customer-managed keys.
- Use private endpoints to secure access between your data and the recovery service vault.
- If you need your backups to be compliant with a regulatory standard, use *Regulatory Compliance in Azure Policy*.
- Use Azure security baselines for Azure Backup (*Azure Security Benchmark*).
- Enable the *soft delete* feature to protect your backups from accidental deletion.
- Consider replicating your backups to another region.

For more information, please refer the following resources:

Security features to help protect hybrid backups that use Azure Backup:

`https://docs.microsoft.com/en-us/azure/backup/backup-azure-security-feature`

Use Azure RBAC to manage Azure backup recovery points:

`https://docs.microsoft.com/en-us/azure/backup/backup-rbac-rs-vault`

Azure Policy Regulatory Compliance controls for Azure Backup:

`https://docs.microsoft.com/en-us/azure/backup/security-controls-policy`

Soft delete for Azure Backup:

`https://docs.microsoft.com/en-us/azure/backup/backup-azure-security-feature-cloud`

Cross Region Restore (**CRR**) for Azure Virtual Machines using Azure Backup:

`https://azure.microsoft.com/en-us/blog/cross-region-restore-crr-for-azure-virtual-machines-using-azure-backup/`

Summary

In this section, we have learned how to securely maintain a VM, based on Azure infrastructure – from logging in to securing network access, troubleshooting using a serial console, patch management, and backup.

Securing Google Compute Engine (GCE) and VM instances

GCE is Google's VM service.

General best practices for Google VMs

Following are some of the best practices to keep in mind:

- Use only trusted images when deploying Google VMs.
- Use a minimal number of packages inside an image, to lower the attack surface.
- Use GCP built-in agents for Google VMs (patch management, hardening, monitoring, and so on).
- For highly sensitive environments, use Google Confidential Computing images, to ensure security and isolation of customers' data.

For more information, please refer the following resources:

List of public images available on GCE:

`https://cloud.google.com/compute/docs/images`

Confidential Computing:

`https://cloud.google.com/confidential-computing`

Best practices for authenticating to a VM instance

Google does not have access to customers' VM instances.

When you run the **create instance** wizard, no credentials are generated.

For Linux instances, you need to manually create a key pair and add the public key to either the instance metadata or the entire GCP project metadata to log in to the machine instance via the SSH protocol.

For more information, please refer the following resource:

Managing SSH keys in metadata:

`https://cloud.google.com/compute/docs/instances/adding-removing-ssh-keys`

For Windows machine instances, you need to manually reset the built-in administrator's password to log in to the machine instance via the RDP protocol.

For more information, please refer the following resource:

Creating passwords for Windows VMs:

`https://cloud.google.com/compute/docs/instances/windows/creating-passwords-for-windows-instances`

The best practices are as follows:

- Keep your private keys in a secured location.
- Avoid storing private keys on a bastion host (machine instances directly exposed to the internet).
- Periodically rotate SSH keys used to access compute instances.
- Periodically review public keys inside the compute instance or GCP project-level SSH key metadata and remove unneeded public keys.
- Join Windows or Linux instances to an AD domain and use your AD credentials to log in to the VMs (and avoid using local credentials or SSH keys completely).

For more information, please refer the following resources:

Quickstart: Joining a Windows VM to a domain:

`https://cloud.google.com/managed-microsoft-ad/docs/quickstart-domain-join-windows`

Quickstart: Joining a Linux VM to a domain:

`https://cloud.google.com/managed-microsoft-ad/docs/quickstart-domain-join-linux`

Best practices for securing network access to a VM instance

Access to GCP resources and services such as VM instances is controlled via **VPC firewall rules**, which are equivalent to the on-premises layer 4 network firewall or access control mechanism.

As a customer, you configure parameters such as the source IP (or CIDR), source service, source tags, destination port (or a predefined protocol), whether the port is TCP or UDP, whether the traffic direction is ingress or egress, and the action to take (either allow or deny).

For remote access and management of Linux machines, limit inbound network access to TCP port `22`.

For remote access and management of Windows machines, limit inbound network access to TCP port `3389`.

The best practices are as follows:

- For remote access protocols (SSH/RDP), limit the source IP (or CIDR) to well-known addresses.
- For file sharing protocols (CIFS/SMB/FTP), limit the source IP (or CIDR) to well-known addresses.
- Set names and descriptions for firewall rules to allow a better understanding of the security group's purpose.
- Use tagging (that is, *labeling*) for firewall rules to allow a better understanding of which firewall rule belongs to which GCP resources.
- Limit the number of ports allowed in a firewall rule to the minimum required ports for allowing your service or application to function.

For more information, please refer the following resource:

Use fewer, broader firewall rule sets when possible:

`https://cloud.google.com/architecture/best-practices-vpc-design#fewer-firewall-rules`

Best practices for securing a serial console connection

For troubleshooting purposes, GCP allows you to connect using a *serial console* (a similar concept to what we used to have in the physical world with network equipment) to resolve network or operating system problems when SSH or RDP connections are not available.

The following command uses the Google Cloud SDK to allow serial access on the entire GCP project:

```
gcloud compute project-info add-metadata \
    --metadata serial-port-enable=TRUE
```

Since this type of remote connectivity exposes your VMs, it is recommended to follow these best practices:

- Configure password-based login to allow users access to the serial console.
- Disable interactive serial console login per compute instance when not required.
- Enable disconnection when the serial console connection is idle.
- Access to the serial console should be limited to the required group of individuals using Google Cloud IAM roles.
- Always set a user password on the target VM instance before allowing access to the serial console.

For more information, please refer the following resource:

Troubleshooting using the serial console:

`https://cloud.google.com/compute/docs/troubleshooting/troubleshooting-using-serial-console`

Best practices for conducting patch management

Patch management is a crucial part of every instance of ongoing maintenance.

To deploy security patches for either Windows- or Linux-based instances, in a standard manner, it is recommended to use Google operating system patch management, using the following method:

1. Deploy the operating system config agent on the target instances.
2. Create a patch job.
3. Run patch deployment.
4. Schedule patch deployment.
5. Review the deployment status inside the operating system patch management dashboard.

The best practices are as follows:

- Use minimal privileges for the accounts using operating system patch management to deploy security patches, according to Google Cloud IAM roles.
- Gradually deploy security patches zone by zone and region by region.
- Use tagging (that is, *labeling*) for your VM instances to allow defining groups of VM instances (for example, *prod* versus *dev* environments).
- For stateless VMs (where no user session data is stored inside a Google VM), replace an existing Google VM with a new instance, created from an up-to-date operating system image.

For more information, please refer the following resources:

Operating system patch management:

`https://cloud.google.com/compute/docs/os-patch-management`

Creating patch jobs:

`https://cloud.google.com/compute/docs/os-patch-management/create-patch-job`

Best practices for operating system updates at scale:

`https://cloud.google.com/blog/products/management-tools/best-practices-for-os-patch-management-on-compute-engine`

Summary

In this section, we have learned how to securely maintain a VM, based on GCP infrastructure – from logging in to securing network access, troubleshooting using the serial console, and patch management.

Securing managed database services

Each cloud provider has its own implementation of managed databases.

According to the shared responsibility model, if we choose to use a managed database, the cloud provider is responsible for the operating system and database layers of the managed database (including patch management, backups, and auditing).

If we have the requirement to deploy a specific build of a database, we can always deploy it inside a VM, but according to the shared responsibility model, we will oversee the entire operating system and database maintenance (including hardening, backup, patch management, and monitoring).

A managed solution for running the database engine – either a common database engine such as MySQL, PostgreSQL, Microsoft SQL Server, an Oracle Database server, or proprietary databases such as Amazon DynamoDB, Azure Cosmos DB, or Google Cloud Spanner, but at the end of the day, the basic idea is the same:

1. Select the database type according to its purpose or use case (relational database, NoSQL database, graph database, in-memory database, and others).
2. Select a database engine (for example, MySQL, PostgreSQL, Microsoft SQL Server, or Oracle Database server).
3. For relational databases, select a machine type (or size) – a ratio between the amount of vCPU and memory, according to their requirements (general-purpose, memory-optimized, and so on).
4. Choose whether high availability is required.
5. Deploy a managed database instance (or cluster).
6. Configure network access control from your cloud environment to your managed database.
7. Enable logging for any access attempt or configuration changes in your managed database.
8. Configure backups on your managed database for recovery purposes.
9. Connect your application to the managed database and begin using the service.

There are various reasons for choosing a managed database solution:

- Maintenance of the database is under the responsibility of the cloud provider.
- Security patch deployment is under the responsibility of the cloud provider.
- Availability of the database is under the responsibility of the cloud provider.
- Backups are included as part of the service (up to a certain amount of storage and amount of backup history).
- Encryption in transit and at rest are embedded as part of a managed solution.
- Auditing is embedded as part of a managed solution.

Since there is a variety of database types and several database engines, in this chapter, we will focus on a single, popular relational database engine – MySQL.

This chapter will not be focusing on non-relational databases.

Next, we are going to see what the best practices are for securing common managed MySQL database services from AWS, Azure, and GCP.

Securing Amazon RDS for MySQL

Amazon **Relational Database Service** (**RDS**) for MySQL is the Amazon-managed MySQL service.

Best practices for configuring IAM for a managed MySQL database service

MySQL supports the following types of authentication methods:

- Local username/password authentication against MySQL's built-in authentication mechanism.
- AWS IAM database authentication.
- AWS Directory Service for Microsoft AD authentication.

The best practices are as follows:

- For the local MySQL master user, create a strong and complex password (at least 15 characters, made up of lowercase and uppercase letters, numbers, and special characters), and keep the password in a secured location.
- For end users who need direct access to the managed database, the preferred method is to use the AWS IAM service, since it allows you to centrally manage all user identities, control their password policy, conduct an audit on their actions (that is, *API calls*), and, in the case of a suspicious security incident, disable the user identity.
- If you manage your user identities using AWS Directory Service for Microsoft AD (AWS-managed Microsoft AD), use this service to authenticate your end users using the Kerberos protocol.

For more information, please refer the following resources:

IAM database authentication for MySQL:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.IAMDBAuth.html`

Using Kerberos authentication for MySQL:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/mysql-kerberos.html`

Best practices for securing network access to a managed MySQL database service

Access to a managed MySQL database service is controlled via *database security groups*, which are equivalent to *security groups* and the on-premises layer 4 network firewall or access control mechanism.

As a customer, you configure parameters such as the source IP (or CIDR) of your web or application servers and the destination IP (or CIDR) of your managed MySQL database service, and AWS configures the port automatically.

The best practices are as follows:

- Managed databases must never be accessible from the internet or a publicly accessible subnet – always use private subnets to deploy your databases.
- Configure security groups for your web or application servers and set the security group as target CIDR when creating a database security group.
- If you need to manage the MySQL database service, either use an EC2 instance (or bastion host) to manage the MySQL database remotely or create a VPN tunnel from your remote machine to the managed MySQL database.
- Since Amazon RDS is a managed service, it is located outside the customer's VPC. An alternative to secure access from your VPC to the managed RDS environment is to use AWS PrivateLink, which avoids sending network traffic outside your VPC, through a secure channel, using an interface VPC endpoint.
- Set names and descriptions for the database security groups to allow a better understanding of the database security group's purpose.
- Use tagging (that is, *labeling*) for database security groups to allow a better understanding of which database security group belongs to which AWS resources.

For more information, please refer the following resources:

Controlling access to RDS with security groups:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.RDSSecurityGroups.html`

Amazon RDS API and interface VPC endpoints (AWS PrivateLink):

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/vpc-interface-endpoints.html`

Best practices for protecting data stored in a managed MySQL database service

A database is meant to store data.

In many cases, a database (and, in this case, a managed MySQL database) may contain sensitive customer data (from a retail store containing customers' data to an organization's sensitive HR data).

To protect customers' data, it is recommended to encrypt data both in transport (when data passes through the network), to avoid detection by an external party, and at rest (data stored inside a database), to avoid data being revealed, even by an internal database administrator.

Encryption allows you to maintain data confidentiality and data integrity (make sure your data is not changed by an untrusted party).

The best practices are as follows:

- Enable SSL/ TLS 1.2 transport layer encryption to your database.
- For non-sensitive environments, encrypt data at rest using AWS KMS (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- For sensitive environments, encrypt data at rest using **customer master key** (**CMK**) management (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).

For more information, please refer the following resources:

Using SSL with a MySQL database instance:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_MySQL.html#MySQL.Concepts.SSLSupport`

Updating applications to connect to MySQL database instances using new SSL/TLS certificates:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/ssl-certificate-rotation-mysql.html`

Select the right encryption options for Amazon RDS database engines:

`https://aws.amazon.com/blogs/database/selecting-the-right-encryption-options-for-amazon-rds-and-amazon-aurora-database-engines/`

CMK management:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.Encryption.Keys.html`

Best practices for conducting auditing and monitoring for a managed MySQL database service

Auditing is a crucial part of data protection.

As with any other managed service, AWS allows you to enable logging and auditing using two built-in services:

- **Amazon CloudWatch**: A service that allows you to log database activities and raise an alarm according to predefined thresholds (for example, a high number of failed logins)
- **AWS CloudTrail**: A service that allows you to monitor API activities (basically, any action performed as part of the AWS RDS API)

The best practices are as follows:

- Enable Amazon CloudWatch alarms for high-performance usage (which may indicate anomalies in the database behavior).
- Enable AWS CloudTrail for any database, to log any activity performed on the database by any user, role, or AWS service.
- Limit access to the CloudTrail logs to the minimum number of employees – preferably in an AWS management account, outside the scope of your end users (including outside the scope of your database administrators), to avoid possible deletion or changes to the audit logs.

For more information, please refer the following resources:

Using Amazon RDS event notifications:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Events.html`

Working with AWS CloudTrail and Amazon RDS:

`https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/logging-using-cloudtrail.html`

Summary

In this section, we have learned how to securely maintain a managed MySQL database, based on AWS infrastructure – from logging in, to securing network access, to data encryption (in transit and at rest), and logging and auditing.

Securing Azure Database for MySQL

Azure Database for MySQL is the Azure-managed MySQL service.

Best practices for configuring IAM for a managed MySQL database service

MySQL supports the following types of authentication methods:

- Local username/password authentication against the MySQL built-in authentication mechanism
- Azure AD authentication

The best practices are as follows:

- For the local MySQL master user, create a strong and complex password (at least 15 characters, made up of lowercase and uppercase letters, numbers, and special characters), and keep the password in a secured location.
- For end users who need direct access to the managed database, the preferred method is to use Azure AD authentication, since it allows you to centrally manage all user identities, control their password policy, conduct an audit on their actions (that is, *API calls*), and, in the case of a suspicious security incident, disable the user identity.

For more information, please refer the following resources:

Use Azure AD for authenticating with MySQL:

`https://docs.microsoft.com/en-us/azure/mysql/concepts-azure-ad-authentication`

Use Azure AD for authentication with MySQL:

`https://docs.microsoft.com/en-us/azure/mysql/howto-configure-sign-in-azure-ad-authentication`

Best practices for securing network access to a managed MySQL database service

Access to a managed MySQL database service is controlled via *firewall rules*, which allows you to configure which IP addresses (or CIDR) are allowed to access your managed MySQL database.

The best practices are as follows:

- Managed databases must never be accessible from the internet or a publicly accessible subnet – always use private subnets to deploy your databases.
- Configure the start IP and end IP of your web or application servers, to limit access to your managed database.
- If you need to manage the MySQL database service, either use an Azure VM (or bastion host) to manage the MySQL database remotely or create a VPN tunnel from your remote machine to the managed MySQL database.
- Since Azure Database for MySQL is a managed service, it is located outside the customer's **virtual network** (**VNet**). An alternative to secure access from your VNet to Azure Database for MySQL is to use a VNet service endpoint, which avoids sending network traffic outside your VNet, through a secure channel.

For more information, please refer the following resources:

Azure Database for MySQL server firewall rules:

`https://docs.microsoft.com/en-us/azure/mysql/concepts-firewall-rules`

Use VNet service endpoints and rules for Azure Database for MySQL:

`https://docs.microsoft.com/en-us/azure/mysql/concepts-data-access-and-security-vnet`

Best practices for protecting data stored in a managed MySQL database service

A database is meant to store data.

In many cases, a database (and, in this case, a managed MySQL database) may contain sensitive customer data (from a retail store containing customers' data to an organization's sensitive HR data).

To protect customers' data, it is recommended to encrypt data both in transport (when the data passes through the network), to avoid detection by an external party, and at rest (data stored inside a database), to avoid data being revealed, even by an internal database administrator.

Encryption allows you to maintain data confidentiality and data integrity (make sure your data is not changed by an untrusted party).

The best practices are as follows:

- Enable TLS 1.2 transport layer encryption to your database.
- For sensitive environments, encrypt data at rest using customer-managed keys stored inside the Azure Key Vault service (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- Keep your customer-managed keys in a secured location for backup purposes.
- Enable the *soft delete* and *purge protection* features on Azure Key Vault to avoid accidental key deletion (which will harm your ability to access your encrypted data).
- Enable auditing on all activities related to encryption keys.

For more information, please refer the following resources:

Azure Database for MySQL data encryption with a customer-managed key:

`https://docs.microsoft.com/en-us/azure/mysql/concepts-data-encryption-mysql`

Azure security baseline for Azure Database for MySQL:

`https://docs.microsoft.com/en-us/security/benchmark/azure/baselines/mysql-security-baseline`

Best practices for conducting auditing and monitoring for a managed MySQL database service

Auditing is a crucial part of data protection.

As with any other managed service, Azure allows you to enable logging and auditing using two built-in services:

- Built-in Azure Database for MySQL audit logs
- Azure Monitor logs

The best practices are as follows:

- Enable audit logs for MySQL.
- Use the Azure Monitor service to detect failed connections.
- Limit access to the Azure Monitor service data to the minimum number of employees to avoid possible deletion or changes to the audit logs.
- Use Advanced Threat Protection for Azure Database for MySQL to detect anomalies or unusual activity in the MySQL database.

For more information, please refer the following resources:

Audit logs in Azure Database for MySQL:

`https://docs.microsoft.com/en-us/azure/mysql/concepts-audit-logs`

Configure and access audit logs for Azure Database for MySQL in the Azure portal:

`https://docs.microsoft.com/en-us/azure/mysql/howto-configure-audit-logs-portal`

Best practices for alerting on metrics with Azure Database for MySQL monitoring:

`https://azure.microsoft.com/en-us/blog/best-practices-for-alerting-on-metrics-with-azure-database-for-mysql-monitoring/`

Security considerations for monitoring data:

`https://docs.microsoft.com/en-us/azure/azure-monitor/roles-permissions-security#security-considerations-for-monitoring-data`

Summary

In this section, we have learned how to securely maintain a managed MySQL database, based on Azure infrastructure – from logging in, to securing network access, to data encryption (in transit and at rest), and logging and auditing.

Securing Google Cloud SQL for MySQL

Google Cloud SQL for MySQL is the Google-managed MySQL service.

Best practices for configuring IAM for a managed MySQL database service

MySQL supports the following types of authentication methods:

- Local username/password authentication against the MySQL built-in authentication mechanism
- Google Cloud IAM authentication

The best practices are as follows:

- For the local MySQL master user, create a strong and complex password (at least 15 characters, made up of lowercase and uppercase letters, numbers, and special characters), and keep the password in a secured location.
- For end users who need direct access to the managed database, the preferred method is to use Google Cloud IAM authentication, since it allows you to centrally manage all user identities, control their password policy, conduct an audit on their actions (that is, *API calls*), and, in the case of a suspicious security incident, disable the user identity.

For more information, please refer the following resources:

Creating and managing MySQL users:

`https://cloud.google.com/sql/docs/mysql/create-manage-users`

MySQL users:

`https://cloud.google.com/sql/docs/mysql/users`

Roles and permissions in Cloud SQL:

`https://cloud.google.com/sql/docs/mysql/roles-and-permissions`

Best practices for securing network access to a managed MySQL database service

Access to a managed MySQL database service is controlled via one of the following options:

- Authorized networks: Allows you to configure which IP addresses (or CIDR) are allowed to access your managed MySQL database
- Cloud SQL Auth proxy: Client installed on your application side, which handles authentication to the Cloud SQL for MySQL database in a secure and encrypted tunnel

The best practices are as follows:

- Managed databases must never be accessible from the internet or a publicly accessible subnet – always use private subnets to deploy your databases.
- If possible, the preferred option is to use the Cloud SQL Auth proxy.
- Configure authorized networks for your web or application servers to allow them access to your Cloud SQL for MySQL.
- If you need to manage the MySQL database service, use either a GCE VM instance to manage the MySQL database remotely or a Cloud VPN (configures an IPSec tunnel to a VPN gateway device).

For more information, please refer the following resources:

Authorizing with authorized networks:

`https://cloud.google.com/sql/docs/mysql/authorize-networks`

Connecting using the Cloud SQL Auth proxy:

`https://cloud.google.com/sql/docs/mysql/connect-admin-proxy`

Cloud VPN overview:

`https://cloud.google.com/network-connectivity/docs/vpn/concepts/overview`

Best practices for protecting data stored in a managed MySQL database service

A database is meant to store data.

In many cases, a database (and, in this case, a managed MySQL database) may contain sensitive customer data (from a retail store containing customer data to an organization's sensitive HR data).

To protect customers' data, it is recommended to encrypt data both in transport (when the data passes through the network), to avoid detection by an external party, and at rest (data stored inside a database), to avoid data being revealed, even by an internal database administrator.

Encryption allows you to maintain data confidentiality and data integrity (make sure your data is not changed by an untrusted party).

The best practices are as follows:

- Enforce TLS 1.2 transport layer encryption on your database.
- For sensitive environments, encrypt data at rest using **customer-managed encryption keys** (**CMEKs**) stored inside the Google Cloud KMS service (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- When using CMEKs, create a dedicated service account, and grant permission to the customers to access the encryption keys inside Google Cloud KMS.
- Enable auditing on all activities related to encryption keys.

For more information, please refer the following resources:

Configuring SSL/TLS certificates:

`https://cloud.google.com/sql/docs/mysql/configure-ssl-instance#enforce-ssl`

Client-side encryption:

`https://cloud.google.com/sql/docs/mysql/client-side-encryption`

Overview of CMEKs:

`https://cloud.google.com/sql/docs/mysql/cmek`

Using CMEKs:

`https://cloud.google.com/sql/docs/mysql/configure-cmek`

Best practices for conducting auditing and monitoring for a managed MySQL database service

Auditing is a crucial part of data protection.

As with any other managed service, GCP allows you to enable logging and auditing using Google Cloud Audit Logs.

The best practices are as follows:

- Admin activity audit logs are enabled by default and cannot be disabled.
- Explicitly enable **data access audit logs** to log activities performed on the database.
- Limit the access to audit logs to the minimum number of employees to avoid possible deletion or changes to the audit logs.

For more information, please refer the following resources:

Audit logs:

`https://cloud.google.com/sql/docs/mysql/audit-logging`

Cloud Audit Logs:

`https://cloud.google.com/logging/docs/audit`

Configuring data access audit logs:

`https://cloud.google.com/logging/docs/audit/configure-data-access`

Permissions and roles:

`https://cloud.google.com/logging/docs/access-control#permissions_and_roles`

Summary

In this section, we have learned how to securely maintain a managed MySQL database, based on GCP infrastructure – from logging in, to securing network access, to data encryption (in transit and at rest), and logging and auditing.

Securing containers

Following VMs, the next evolution in the compute era is containers.

Containers behave like VMs, but with a much smaller footprint.

Instead of having to deploy an application above an entire operating system, you could use containers to deploy your required application, with only the minimum required operating system libraries and binaries.

Containers have the following benefits over VMs:

- **Small footprint**: Only required libraries and binaries are stored inside a container.
- **Portability**: You can develop an application inside a container on your laptop and run it at a large scale in a production environment with hundreds or thousands of container instances.
- Fast deployment and updates compared to VMs.

The following diagram presents the architectural differences between VMs and containers:

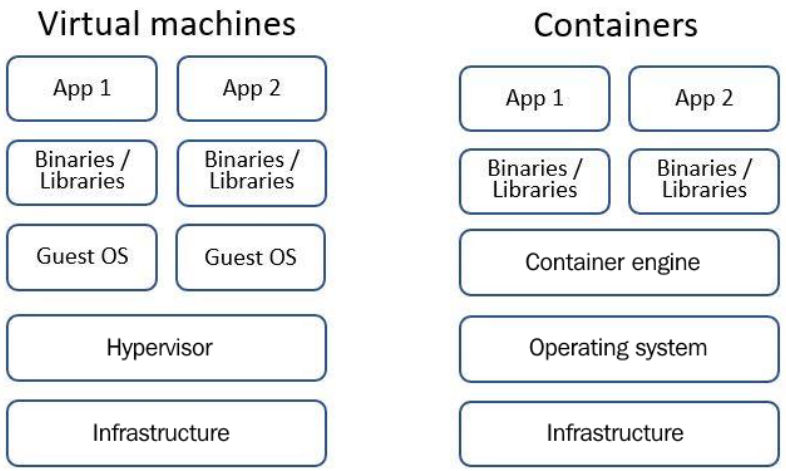

Figure 2.1 – VMs versus containers

If you are still in the development phase, you can install a container engine on your laptop and create a new container (or download an existing container) locally, until you complete the development phase.

When you move to production and have a requirement to run hundreds of container instances, you need an orchestrator – a mechanism (or a managed service) for managing container deployment, health check monitoring, container recycling, and more.

Docker was adopted by the industry as a de facto standard for wrapping containers, and in the past couple of years, more and more cloud vendors have begun to support a new initiative for wrapping containers called the **Open Container Initiative** (**OCI**).

Kubernetes is an open source project (developed initially by Google) and is now considered the industry de facto standard for orchestrating, deploying, scaling, and managing containers.

In this section, I will present the most common container orchestrators available as managed services.

For more information, please refer the following resources:

What is a container?

`https://www.docker.com/resources/what-container`

Open Container Initiative:

`https://opencontainers.org/`

OCI artifact support in Amazon ECR:

`https://aws.amazon.com/blogs/containers/oci-artifact-support-in-amazon-ecr/`

Azure and OCI images:

`https://docs.microsoft.com/en-us/azure/container-registry/container-registry-image-formats#oci-images`

GCP and OCI image format:

`https://cloud.google.com/artifact-registry/docs/supported-formats#oci`

The Kubernetes project:

`https://kubernetes.io/`

Next, we are going to see what the best practices are for securing common container and Kubernetes services from AWS, Azure, and GCP.

Securing Amazon Elastic Container Service (ECS)

ECS is the Amazon-managed container orchestration service.

It can integrate with other AWS services such as Amazon **Elastic Container Registry** (**ECR**) for storing containers, AWS IAM for managing permissions to ECS, and Amazon CloudWatch for monitoring ECS.

Best practices for configuring IAM for Amazon ECS

AWS IAM is the supported service for managing permissions to access and run containers through Amazon ECS.

The best practices are as follows:

- Grant minimal IAM permissions for the Amazon ECS service (for running tasks, accessing S3 buckets, monitoring using CloudWatch Events, and so on).
- If you are managing multiple AWS accounts, use temporary credentials (using AWS Security Token Service or the *AssumeRole* capability) to manage ECS on the target AWS account with credentials from a source AWS account.
- Use service roles to allow the ECS service to assume your role and access resources such as S3 buckets, RDS databases, and so on.
- Use IAM roles to control access to Amazon **Elastic File System** (**EFS**) from ECS.
- Enforce **multi-factor authentication** (**MFA**) for end users who have access to the AWS console and perform privileged actions such as managing the ECS service.
- Enforce policy conditions such as requiring end users to connect to the ECS service using a secured channel (SSL/TLS), connecting using MFA, log in at specific hours of the day, and so on.
- Store your container images inside Amazon ECR and grant minimal IAM permissions for accessing and managing Amazon ECR.

For more information, please refer the following resources:

Amazon ECS container instance IAM role:

`https://docs.aws.amazon.com/AmazonECS/latest/developerguide/instance_IAM_role.html`

IAM roles for tasks:

`https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-iam-roles.html`

Authorization based on Amazon ECS tags:

`https://docs.aws.amazon.com/AmazonECS/latest/userguide/security_iam_service-with-iam.html#security_iam_service-with-iam-tags`

Using IAM to control filesystem data access:

`https://docs.aws.amazon.com/efs/latest/ug/iam-access-control-nfs-efs.html`

Amazon ECS task and container security:

`https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/security-tasks-containers.html`

Best practices for securing network access to Amazon ECS

Since Amazon ECS is a managed service, it is located outside the customer's VPC. An alternative to secure access from your VPC to the managed ECS environment is to use AWS PrivateLink, which avoids sending network traffic outside your VPC, through a secure channel, using an interface VPC endpoint.

The best practices are as follows:

- Use a secured channel (TLS 1.2) to control Amazon ECS using API calls.
- Use VPC security groups to allow access from your VPC to the Amazon ECS VPC endpoint.
- If you use AWS Secrets Manager to store sensitive data (such as credentials) from Amazon ECS, use a Secrets Manager VPC endpoint when configuring security groups.
- If you use AWS Systems Manager to remotely execute commands on Amazon ECS, use Systems Manager VPC endpoints when configuring security groups.
- Store your container images inside Amazon ECR and for non-sensitive environments, encrypt your container images inside Amazon ECR using AWS KMS (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- For sensitive environments, encrypt your container images inside Amazon ECR using CMK management (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- If you use Amazon ECR to store your container images, use VPC security groups to allow access from your VPC to the Amazon ECR interface's VPC endpoint.

For more information, please refer the following resource:

Amazon ECS interface VPC endpoints (AWS PrivateLink):

`https://docs.aws.amazon.com/AmazonECS/latest/developerguide/vpc-endpoints.html`

Best practices for conducting auditing and monitoring in Amazon ECS

Auditing is a crucial part of data protection.

As with any other managed service, AWS allows you to enable logging and auditing using two built-in services:

- **Amazon CloudWatch**: A service that allows you to log containers' activities and raise an alarm according to predefined thresholds (for example, low memory resources or high CPU, which requires up-scaling your ECS cluster)
- **AWS CloudTrail**: A service that allows you to monitor API activities (basically, any action performed on the ECS cluster)

The best practices are as follows:

- Enable Amazon CloudWatch alarms for high-performance usage (which may indicate an anomaly in the ECS cluster behavior).
- Enable AWS CloudTrail for any action performed on the ECS cluster.
- Limit the access to the CloudTrail logs to the minimum number of employees – preferably in an AWS management account, outside the scope of your end users (including outside the scope of your ECS cluster administrators), to avoid possible deletion or changes to the audit logs.

For more information, please refer the following resources:

Logging and monitoring in Amazon ECS:

`https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-logging-monitoring.html`

Logging Amazon ECS API calls with AWS CloudTrail:

`https://docs.aws.amazon.com/AmazonECS/latest/developerguide/logging-using-cloudtrail.html`

Best practices for enabling compliance on Amazon ECS

Security configuration is a crucial part of your infrastructure.

Amazon allows you to conduct ongoing compliance checks against well-known security standards (such as the Center for Internet Security Benchmarks).

The best practices are as follows:

- Use only trusted image containers and store them inside Amazon ECR – a private repository for storing your organizational images.
- Run the *Docker Bench for Security* tool on a regular basis to check for compliance with CIS Benchmarks for Docker containers.
- Build your container images from scratch (to avoid malicious code in preconfigured third-party images).
- Scan your container images for vulnerabilities in libraries and binaries and update your images on a regular basis.
- Configure your images with a read-only root filesystem to avoid unintended upload of malicious code into your images.

For more information, please refer the following resources:

Docker Bench for Security:

`https://github.com/docker/docker-bench-security`

Amazon ECR private repositories:

`https://docs.aws.amazon.com/AmazonECR/latest/userguide/Repositories.html`

Summary

In this section, we have learned how to securely maintain Amazon ECS, based on AWS infrastructure – from logging in, to securing network access, to logging and auditing, and security compliance.

Securing Amazon Elastic Kubernetes Service (EKS)

EKS is the Amazon-managed Kubernetes orchestration service.

It can integrate with other AWS services, such as Amazon ECR for storing containers, AWS IAM for managing permissions to EKS, and Amazon CloudWatch for monitoring EKS.

Best practices for configuring IAM for Amazon EKS

AWS IAM is the supported service for managing permissions to access and run containers through Amazon EKS.

The best practices are as follows:

- Grant minimal IAM permissions for accessing and managing Amazon EKS.
- If you are managing multiple AWS accounts, use temporary credentials (using AWS Security Token Service or the *AssumeRole* capability) to manage EKS on the target AWS account with credentials from a source AWS account.
- Use service roles to allow the EKS service to assume your role and access resources such as S3 buckets and RDS databases.
- For authentication purposes, avoid using service account tokens.
- Create an IAM role for each newly created EKS cluster.
- Create a service account for each newly created application.
- Always run applications using a non-root user.
- Use IAM roles to control access to storage services (such as Amazon EBS, Amazon EFS, and Amazon FSx for Lustre) from EKS.
- Enforce MFA for end users who have access to the AWS console and perform privileged actions such as managing the EKS service.
- Store your container images inside Amazon ECR and grant minimal IAM permissions for accessing and managing Amazon ECR.

For more information, please refer the following resources:

How Amazon EKS works with IAM:

`https://docs.aws.amazon.com/eks/latest/userguide/security_iam_service-with-iam.html`

IAM roles for service accounts:

`https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html`

IAM:

`https://aws.github.io/aws-eks-best-practices/security/docs/iam/`

Best practices for securing network access to Amazon EKS

Since Amazon EKS is a managed service, it is located outside the customer's VPC. An alternative to secure access from your VPC to the managed EKS environment is to use AWS PrivateLink, which avoids sending network traffic outside your VPC, through a secure channel, using an interface VPC endpoint.

The best practices are as follows:

- Use TLS 1.2 to control Amazon EKS using API calls.
- Use TLS 1.2 when configuring Amazon EKS behind AWS Application Load Balancer or AWS Network Load Balancer.
- Use TLS 1.2 between your EKS control plane and the EKS cluster's worker nodes.
- Use VPC security groups to allow access from your VPC to the Amazon EKS VPC endpoint.
- Use VPC security groups between your EKS control plane and the EKS cluster's worker nodes.
- Use VPC security groups to protect access to your EKS Pods.
- Disable public access to your EKS API server – either use an EC2 instance (or bastion host) to manage the EKS cluster remotely or create a VPN tunnel from your remote machine to your EKS cluster.
- If you use AWS Secrets Manager to store sensitive data (such as credentials) from Amazon EKS, use a Secrets Manager VPC endpoint when configuring security groups.
- Store your container images inside Amazon ECR, and for non-sensitive environments, encrypt your container images inside Amazon ECR using AWS KMS (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- For sensitive environments, encrypt your container images inside Amazon ECR using CMK management (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- If you use Amazon ECR to store your container images, use VPC security groups to allow access from your VPC to the Amazon ECR interface VPC endpoint.

For more information, please refer the following resources:

Network security:

`https://aws.github.io/aws-eks-best-practices/security/docs/network`

Amazon EKS networking:

`https://docs.aws.amazon.com/eks/latest/userguide/eks-networking.html`

Introducing security groups for Pods:

`https://aws.amazon.com/blogs/containers/introducing-security-groups-for-pods/`

EKS best practice guides:

`https://aws.github.io/aws-eks-best-practices/`

Best practices for conducting auditing and monitoring in Amazon EKS

Auditing is a crucial part of data protection.

As with any other managed service, AWS allows you to enable logging and auditing using two built-in services:

- **Amazon CloudWatch**: A service that allows you to log EKS cluster activities and raise an alarm according to predefined thresholds (for example, low memory resources or high CPU, which requires up-scaling your EKS cluster).
- **AWS CloudTrail**: A service that allows you to monitor API activities (basically, any action performed on the EKS cluster).

The best practices are as follows:

- Enable the Amazon EKS control plane when logging in to Amazon CloudWatch – this allows you to log API calls, audit, and authentication information from your EKS cluster.
- Enable AWS CloudTrail for any action performed on the EKS cluster.
- Limit the access to the CloudTrail logs to the minimum number of employees – preferably in an AWS management account, outside the scope of your end users (including outside the scope of your EKS cluster administrators), to avoid possible deletion or changes to the audit logs.

For more information, please refer the following resources:

Amazon EKS control plane logging:

`https://docs.aws.amazon.com/eks/latest/userguide/control-plane-logs.html`

Auditing and logging:

`https://aws.github.io/aws-eks-best-practices/security/docs/detective/`

Best practices for enabling compliance on Amazon EKS

Security configuration is a crucial part of your infrastructure.

Amazon allows you to conduct ongoing compliance checks against well-known security standards (such as CIS Benchmarks).

The best practices are as follows:

- Use only trusted image containers and store them inside Amazon ECR – a private repository for storing your organizational images.
- Run the `kube-bench` tool on a regular basis to check for compliance with CIS Benchmarks for Kubernetes.
- Run the *Docker Bench for Security* tool on a regular basis to check for compliance with CIS Benchmarks for Docker containers.
- Build your container images from scratch (to avoid malicious code in preconfigured third-party images).
- Scan your container images for vulnerabilities in libraries and binaries and update your images on a regular basis.
- Configure your images with a read-only root filesystem to avoid unintended upload of malicious code into your images.

For more information, please refer the following resources:

Configuration and vulnerability analysis in Amazon EKS:

`https://docs.aws.amazon.com/eks/latest/userguide/configuration-vulnerability-analysis.html`

Introducing the CIS Amazon EKS Benchmark:

`https://aws.amazon.com/blogs/containers/introducing-cis-amazon-eks-benchmark/`

Compliance:

`https://aws.github.io/aws-eks-best-practices/security/docs/compliance/`

Image security:

`https://aws.github.io/aws-eks-best-practices/security/docs/image/`

Pod security:

`https://aws.github.io/aws-eks-best-practices/security/docs/pods/`

`kube-bench`:

`https://github.com/aquasecurity/kube-bench`

Docker Bench for Security:

`https://github.com/docker/docker-bench-security`

Amazon ECR private repositories:

`https://docs.aws.amazon.com/AmazonECR/latest/userguide/Repositories.html`

Summary

In this section, we have learned how to securely maintain Amazon EKS, based on AWS infrastructure – from logging in, to securing network access, to logging and auditing, and security compliance.

Securing Azure Container Instances (ACI)

ACI is the Azure-managed container orchestration service.

It can integrate with other Azure services, such as **Azure Container Registry** (**ACR**) for storing containers, Azure AD for managing permissions to ACI, Azure Files for persistent storage, and Azure Monitor.

Best practices for configuring IAM for ACI

Although ACI does not have its own authentication mechanism, it is recommended to use ACR to store your container images in a private registry.

ACR supports the following authentication methods:

- **Managed identity**: A user or system account from Azure AD
- **Service principal**: An application, service, or platform that needs access to ACR

The best practices are as follows:

- Grant minimal permissions for accessing and managing ACR, using Azure RBAC.
- When passing sensitive information (such as credential secrets), make sure the traffic is encrypted in transit through a secure channel (TLS).
- If you need to store sensitive information (such as credentials), store it inside the Azure Key Vault service.
- For sensitive environments, encrypt information (such as credentials) using *customer-managed keys*, stored inside the Azure Key Vault service.
- Disable the ACR built-in admin user.

For more information, please refer the following resources:

Authenticate with ACR:

```
https://docs.microsoft.com/en-us/azure/container-registry/
container-registry-authentication
```

ACR roles and permissions:

```
https://docs.microsoft.com/en-us/azure/container-registry/
container-registry-roles
```

Encrypt a registry using a customer-managed key:

```
https://docs.microsoft.com/en-us/azure/container-registry/
container-registry-customer-managed-keys
```

Best practices for conducting auditing and monitoring in ACI

Auditing is a crucial part of data protection.

As with any other managed service, Azure allows you to enable logging and auditing using Azure Monitor for containers – a service that allows you to log container-related activities and raise an alarm according to predefined thresholds (for example, low memory resources or high CPU, which requires up-scaling your container environment).

The best practices are as follows:

- Enable audit logging for Azure resources, using Azure Monitor, to log authentication-related activities of your ACR.
- Limit the access to the Azure Monitor logs to the minimum number of employees to avoid possible deletion or changes to the audit logs.

For more information, please refer the following resources:

Container insights overview:

`https://docs.microsoft.com/en-us/azure/azure-monitor/containers/container-insights-overview`

Container monitoring solution in Azure Monitor:

`https://docs.microsoft.com/en-us/azure/azure-monitor/containers/containers`

Best practices for enabling compliance on ACI

Security configuration is a crucial part of your infrastructure.

Azure allows you to conduct ongoing compliance checks against well-known security standards (such as CIS Benchmarks).

The best practices are as follows:

- Use only trusted image containers and store them inside ACR – a private repository for storing your organizational images.
- Integrate ACR with Azure Security Center, to detect non-compliant images (from the CIS standard).
- Build your container images from scratch (to avoid malicious code in preconfigured third-party images).
- Scan your container images for vulnerabilities in libraries and binaries and update your images on a regular basis.

For more information, please refer the following resources:

Azure security baseline for ACI:

```
https://docs.microsoft.com/en-us/security/benchmark/azure/
baselines/container-instances-security-baseline
```

Azure security baseline for ACR:

```
https://docs.microsoft.com/en-us/security/benchmark/azure/
baselines/container-registry-security-baseline
```

Security considerations for ACI:

```
https://docs.microsoft.com/en-us/azure/container-instances/
container-instances-image-security
```

Introduction to private Docker container registries in Azure:

```
https://docs.microsoft.com/en-us/azure/container-registry/
container-registry-intro
```

Summary

In this section, we have learned how to securely maintain ACI, based on Azure infrastructure – from logging in to auditing and monitoring and security compliance.

Securing Azure Kubernetes Service (AKS)

AKS is the Azure-managed Kubernetes orchestration service.

It can integrate with other Azure services, such as ACR for storing containers, Azure AD for managing permissions to AKS, Azure Files for persistent storage, and Azure Monitor.

Best practices for configuring IAM for Azure AKS

Azure AD is the supported service for managing permissions to access and run containers through Azure AKS.

The best practices are as follows:

- Enable Azure AD integration for any newly created AKS cluster.
- Grant minimal permissions for accessing and managing AKS, using Azure RBAC.
- Grant minimal permissions for accessing and managing ACR, using Azure RBAC.
- Create a unique service principal for each newly created AKS cluster.

- When passing sensitive information (such as credential secrets), make sure the traffic is encrypted in transit through a secure channel (TLS).
- If you need to store sensitive information (such as credentials), store it inside the Azure Key Vault service.
- For sensitive environments, encrypt information (such as credentials) using *customer-managed keys*, stored inside the Azure Key Vault service.

For more information, please refer the following resources:

AKS-managed Azure AD integration:

`https://docs.microsoft.com/en-us/azure/aks/managed-aad`

Best practices for authentication and authorization in AKS:

`https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-identity`

Best practices for securing network access to Azure AKS

Azure AKS exposes services to the internet – for that reason, it is important to plan before deploying each Azure AKS cluster.

The best practices are as follows:

- Avoid exposing the AKS cluster control plane (API server) to the public internet – create a private cluster with an internal IP address and use authorized IP ranges to define which IPs can access your API server.
- Use the Azure Firewall service to restrict outbound traffic from AKS cluster nodes to external DNS addresses (for example, software updates from external sources).
- Use TLS 1.2 to control Azure AKS using API calls.
- Use TLS 1.2 when configuring Azure AKS behind Azure Load Balancer.
- Use TLS 1.2 between your AKS control plane and the AKS cluster's nodes.
- For small AKS deployments, use the `kubenet` plugin to implement network policies and protect the AKS cluster.
- For large production deployments, use the Azure CNI Kubernetes plugin to implement network policies and protect the AKS cluster.
- Use Azure network security groups to block SSH traffic to the AKS cluster nodes, from the AKS subnets only.

- Use network policies to protect the access between the Kubernetes Pods.
- Disable public access to your AKS API server – either use an Azure VM (or Azure Bastion) to manage the AKS cluster remotely or create a VPN tunnel from your remote machine to your AKS cluster.
- Disable or remove the HTTP application routing add-on.
- If you need to store sensitive information (such as credentials), store it inside the Azure Key Vault service.
- For sensitive environments, encrypt information (such as credentials) using *customer-managed keys*, stored inside the Azure Key Vault service.

For more information, please refer the following resources:

Best practices for network connectivity and security in AKS:

`https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-network`

Best practices for cluster isolation in AKS:

`https://docs.microsoft.com/en-us/azure/aks/operator-best-practices-cluster-isolation`

Create a private AKS cluster:

`https://docs.microsoft.com/en-us/azure/aks/private-clusters`

Best practices for conducting auditing and monitoring in Azure AKS

Auditing is a crucial part of data protection.

As with any other managed service, Azure allows you to enable logging and auditing using Azure Monitor for containers – a service that allows you to log container-related activities and raise an alarm according to predefined thresholds (for example, low memory resources or high CPU, which requires up-scaling your container environment).

The best practices are as follows:

- Enable audit logging for Azure resources, using Azure Monitor, to log authentication-related activities of your ACR.
- Limit the access to the Azure Monitor logs to the minimum number of employees to avoid possible deletion or changes to the audit logs.

For more information, please refer the following resources:

ACI overview:

```
https://docs.microsoft.com/en-us/azure/azure-monitor/
containers/container-insights-overview
```

Container monitoring solution in Azure Monitor:

```
https://docs.microsoft.com/en-us/azure/azure-monitor/
containers/containers
```

Best practices for enabling compliance on Azure AKS

Security configuration is a crucial part of your infrastructure.

Azure allows you to conduct ongoing compliance checks against well-known security standards (such as CIS Benchmarks).

The best practices are as follows:

- Use only trusted image containers and store them inside ACR – a private repository for storing your organizational images.
- Use Azure Defender for Kubernetes to protect Kubernetes clusters from vulnerabilities.
- Use Azure Defender for container registries to detect and remediate vulnerabilities in container images.
- Integrated Azure Container Registry with Azure Security Center to detect non-compliant images (from the CIS standard).
- Build your container images from scratch (to avoid malicious code in preconfigured third-party images).
- Scan your container images for vulnerabilities in libraries and binaries and update your images on a regular basis.

For more information, please refer the following resources:

Introduction to Azure Defender for Kubernetes:

```
https://docs.microsoft.com/en-us/azure/security-center/
defender-for-kubernetes-introduction
```

Use Azure Defender for container registries to scan your images for vulnerabilities:

```
https://docs.microsoft.com/en-us/azure/security-center/
defender-for-container-registries-usage
```

Azure security baseline for ACI:

```
https://docs.microsoft.com/en-us/security/benchmark/azure/
baselines/container-instances-security-baseline
```

Azure security baseline for ACR:

```
https://docs.microsoft.com/en-us/security/benchmark/azure/
baselines/container-registry-security-baseline
```

Security considerations for ACI:

```
https://docs.microsoft.com/en-us/azure/container-instances/
container-instances-image-security
```

Introduction to private Docker container registries in Azure:

```
https://docs.microsoft.com/en-us/azure/container-registry/
container-registry-intro
```

Summary

In this section, we have learned how to securely maintain AKS, based on Azure infrastructure – from logging in, to network access, to auditing and monitoring, and security compliance.

Securing Google Kubernetes Engine (GKE)

GKE is the Google-managed Kubernetes orchestration service.

It can integrate with other GCP services, such as Google Container Registry for storing containers, Google Cloud IAM for managing permissions to GKE, Google Filestore for persistent storage, and Google Cloud operations for monitoring.

Best practices for configuring IAM for GKE

Google Cloud IAM is the supported service for managing permissions to access and run containers through GKE.

The best practices are as follows:

- Grant minimal permissions for accessing and managing the Google Cloud IAM service, using Kubernetes RBAC.
- Use Google Groups to manage permissions to your GKE cluster.
- Use the Google IAM recommender to set the minimal permissions for your GKE cluster.
- Create a unique service account with minimal permissions for any newly created GKE cluster.
- Enforce the use of MFA for any user who needs access to manage your GKE cluster.
- If you need to store sensitive information (such as credentials), store it inside the Google Cloud KMS service.
- For sensitive environments, encrypt information (such as credentials) using CMEKs stored inside the Google Cloud KMS service.

For more information, please refer the following resources:

Creating IAM policies:

`https://cloud.google.com/kubernetes-engine/docs/how-to/iam`

Configuring RBAC:

`https://cloud.google.com/kubernetes-engine/docs/how-to/role-based-access-control`

Enforce least privilege with recommendations:

`https://cloud.google.com/iam/docs/recommender-overview`

Use least privilege Google service accounts:

`https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster#permissions`

Secret management:

`https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster#secret_management`

Best practices for securing network access to GKE

GKE exposes services to the internet – for that reason, it is important to plan before deploying each GKE cluster.

The best practices are as follows:

- Create private GKE clusters to avoid exposing the GKE cluster control plane (API server) to the public internet – use alias IP ranges to configure which IPs can access your GKE cluster.
- Use authorized networks to configure who can access your GKE cluster control plane.
- Use VPC-native networking to protect the access between the Kubernetes Pods.
- Use network policies for Kubernetes to protect the access between the Kubernetes Pods.
- Use shielded GKE nodes as an additional layer of protection to your GKE cluster nodes.
- Create separate namespaces for your applications, according to RBAC requirements.
- Enable a GKE sandbox to achieve better isolation of your GKE cluster Pods.
- Use TLS 1.2 to control your GKE cluster using API calls.
- Use TLS 1.2 between your GKE control plane and the GKE cluster's nodes.
- Use TLS 1.2 when configuring the GKE cluster behind Google Load Balancer.
- Disable public access to your GKE cluster API server – use a Google VM (or a Bastion host) to manage the GKE cluster remotely.

For more information, please refer the following resources:

Creating a private cluster:

```
https://cloud.google.com/kubernetes-engine/docs/how-to/
private-clusters
```

Restrict network access to the control plane and nodes:

```
https://cloud.google.com/kubernetes-engine/docs/how-to/
hardening-your-cluster#restrict_network_access_to_the_control_
plane_and_nodes
```

Restrict traffic among Pods with a network policy:

`https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster#restrict_with_network_policy`

Network security:

`https://cloud.google.com/kubernetes-engine/docs/concepts/security-overview#network_security`

Harden workload isolation with a GKE sandbox:

`https://cloud.google.com/kubernetes-engine/docs/how-to/sandbox-pods`

Best practices for conducting auditing and monitoring in GKE

Auditing is a crucial part of data protection.

As with any other managed service, Google allows you to enable logging and auditing using the Google Cloud Logging service – a service that allows you to audit container-related activities.

The best practices are as follows:

- Enable logging for any newly created GKE cluster and integrate the item with the Google Cloud Logging service, to log all audit activities related to your GKE cluster.
- When using a *container-optimized operating system* image, make sure you send its Linux audit logs to the Google Cloud Logging service.
- Limit the access to the Google Cloud Logging service logs to the minimum number of employees to avoid possible deletion or changes to the audit logs.

For more information, please refer the following resources:

Audit policy:

`https://cloud.google.com/kubernetes-engine/docs/concepts/audit-policy`

Overview of Google Cloud's operations suite for GKE:

`https://cloud.google.com/stackdriver/docs/solutions/gke`

Remediating security health analytics findings:

```
https://cloud.google.com/security-command-center/docs/how-to-
remediate-security-health-analytics-findings#container_
vulnerability_findings
```

Best practices for enabling compliance in GKE

Security configuration is a crucial part of your infrastructure.

Google allows you to conduct ongoing compliance checks against well-known security standards (such as CIS Benchmarks).

The best practices are as follows:

- Use only trusted image containers and store them inside Google Container Registry – a private repository for storing your organizational images.
- Always use the latest build of Kubernetes on both your GKE cluster and cluster nodes.
- Use the GKE auditor to detect GKE misconfigurations.
- Use container-optimized operating systems when creating new container images, for better security.
- Build your container images from scratch (to avoid malicious code in preconfigured third-party images).
- Scan your container images for vulnerabilities in libraries and binaries and update your images on a regular basis.
- Use Google Binary Authorization to make sure you use only signed container images from trusted authorities.
- Use container threat detection to detect attacks against your container images in real time.

For more information, please refer the following resources:

Container threat detection conceptual overview:

`https://cloud.google.com/security-command-center/docs/concepts-container-threat-detection-overview`

GKE CIS 1.1.0 Benchmark Inspec Profile:

`https://github.com/GoogleCloudPlatform/inspec-gke-cis-benchmark`

GKE auditor:

`https://github.com/google/gke-auditor`

Node images:

`https://cloud.google.com/kubernetes-engine/docs/concepts/node-images#containerd_node_images`

Binary authorization:

`https://cloud.google.com/binary-authorization`

Container Registry:

`https://cloud.google.com/container-registry`

Summary

In this section, we have learned how to securely maintain the Google Kubernetes service, based on GCP infrastructure – from logging in, to network access, to auditing and monitoring, and security compliance.

Securing serverless/function as a service

Although the name implies that there are no servers, the term *serverless* or *function as a service* means that you, as a customer of the service, are not in charge of the underlying compute infrastructure (operating system maintenance, scale, runtime management, and so on) – you simply import your code (according to the supported language by each cloud provider), select your preferred runtime, select the amount of required memory per function (which affects the amount of CPU), and set the trigger to invoke the function.

The following diagram presents the architectural differences between VMs, containers, and serverless:

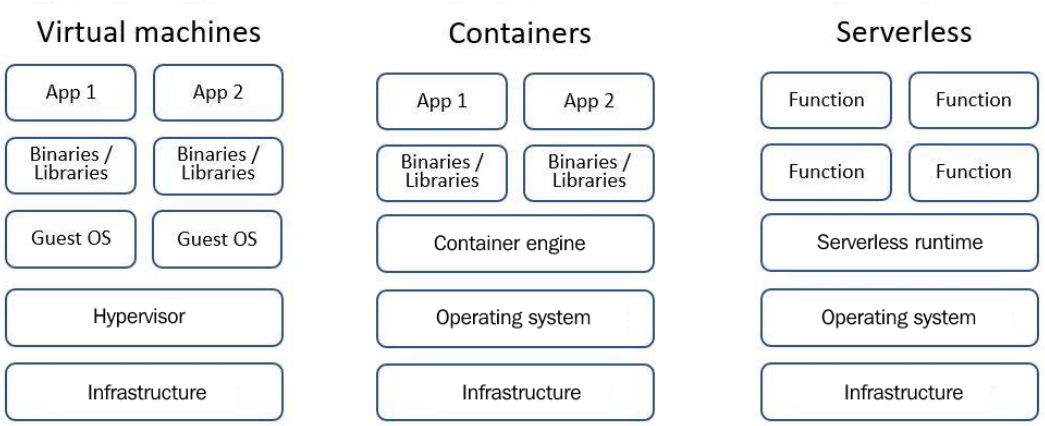

Figure 2.2 – VMs versus containers versus serverless

In this section, I will present the most common serverless/function as a service platforms.

Then, we are going to see what the best practices are for securing common serverless services from AWS, Azure, and GCP.

Securing AWS Lambda

AWS Lambda is the Amazon serverless service.

It can integrate with other AWS services, such as AWS IAM for managing permissions to AWS Lambda, Amazon CloudWatch for monitoring AWS Lambda, and Amazon S3 and Amazon EFS for persistent storage.

Best practices for configuring IAM for AWS Lambda

AWS IAM is the supported service for managing permissions to AWS Lambda.

The best practices are as follows:

- Grant minimal IAM permissions for any newly created AWS Lambda function (for running tasks, accessing S3 buckets, monitoring using CloudWatch Events, and so on) – match a specific IAM role to any newly created AWS Lambda function.
- Use open source tools such as `serverless-puresec-cli` to generate IAM roles for your function.
- Avoid storing credentials inside AWS Lambda code.

- If you need to store sensitive data (such as credentials), use AWS Secrets Manager.
- For better protection of your Lamba functions, configure AWS Lambda behind Amazon API Gateway.
- For sensitive environments, encrypt Lambda environment variables using CMK management (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).
- Use TLS 1.2 to encrypt sensitive data over the network.
- Enforce MFA for end users who have access to the AWS API (console, CLI, and SDK) and perform privileged actions such as managing the Lambda service.

For more information, please refer the following resources:

Identity-based IAM policies for Lambda:

`https://docs.aws.amazon.com/lambda/latest/dg/access-control-identity-based.html`

Security best practices:

`https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/security-best-practices.html`

Encrypting Lambda environment variables:

`https://docs.aws.amazon.com/whitepapers/latest/kms-best-practices/encrypting-lambda-environment-variables.html`

`serverless-puresec-cli`:

`https://github.com/puresec/serverless-puresec-cli`

Best practices for securing network access to AWS Lambda

AWS Lambda can be deployed either as an external resource outside your VPC or inside your VPC – for that reason, it is important to plan before deploying each Lambda function.

The best practices are as follows:

- Use Amazon API Gateway to restrict access to your Lambda function, from a specific IP address or CIDR.

- If your Lambda function is located outside a VPC, and the Lambda function needs access to resources inside your VPC, use AWS PrivateLink, which avoids sending network traffic outside your VPC, through a secure channel, using an interface VPC endpoint.
- If your Lambda function is located inside your VPC, and the Lambda function needs access to external resources on the internet, use the NAT gateway to give your Lambda function the required access, without exposing Lambda to the internet directly.
- Use TLS 1.2 to encrypt traffic to and from your Lambda functions.

For more information, please refer the following resources:

Data protection in AWS Lambda:

`https://docs.aws.amazon.com/lambda/latest/dg/security-dataprotection.html`

AWS Lambda now supports AWS PrivateLink:

`https://aws.amazon.com/about-aws/whats-new/2020/10/aws-lambda-now-supports-aws-privatelink/`

Configuring a Lambda function to access resources in a VPC:

`https://docs.aws.amazon.com/lambda/latest/dg/configuration-vpc.html`

How do I give internet access to a Lambda function that's connected to an Amazon VPC?

`https://aws.amazon.com/premiumsupport/knowledge-center/internet-access-lambda-function/`

Best practices for conducting auditing and monitoring in AWS Lambda

Auditing is a crucial part of data protection.

As with any other managed service, AWS allows you to enable auditing using the AWS CloudTrail service – a service that allows you to audit API-related activities.

The best practices are as follows:

- Enable enhanced monitoring of your Lambda functions.
- Use Amazon CloudWatch to detect spikes in Lambda usage.
- Use AWS CloudTrail to monitor API activities related to your Lambda function.

For more information, please refer the following resources:

Using AWS Lambda with AWS CloudTrail:

`https://docs.aws.amazon.com/lambda/latest/dg/with-cloudtrail.html`

Using AWS Lambda with Amazon CloudWatch Events:

`https://docs.aws.amazon.com/lambda/latest/dg/services-cloudwatchevents.html`

Best practices for conducting compliance, configuration change, and secure coding in AWS Lambda

Serverless, or function as a service, is mainly code running inside a closed managed environment.

As a customer, you cannot control the underlying infrastructure – as a result, you must invest in secure coding to avoid attackers breaking into your application and causing harm that AWS cannot protect.

The best practices are as follows:

- Follow the *OWASP Serverless Top 10* project documentation when writing your Lambda function code.
- Enable versions in your Lambda functions, to be able to roll back to previous code.
- Use AWS Signer to sign your Lambda function code and make sure you only run signed code.
- If you use Amazon API Gateway in front of your Lambda functions, use the API Gateway Lambda authorizer as an extra layer of protection for authorizing access to your Lambda functions.
- Use AWS Config to check for changes in your Lambda functions.
- Use Amazon Inspector assessment templates to detect non-compliance or the use of old versions of a runtime in your Lambda functions.

For more information, please refer the following resources:

Using AWS Lambda with AWS Config:

`https://docs.aws.amazon.com/lambda/latest/dg/services-config.html`

Lambda function versions:

`https://docs.aws.amazon.com/lambda/latest/dg/configuration-versions.html`

Use API Gateway Lambda authorizers:

`https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html`

Setting up automatic assessment runs through a Lambda function:

`https://docs.aws.amazon.com/inspector/latest/userguide/inspector_assessments.html#assessment_runs-schedule`

Configuring code signing for AWS Lambda:

`https://docs.aws.amazon.com/lambda/latest/dg/configuration-codesigning.html`

OWASP Serverless Top 10:

`https://owasp.org/www-project-serverless-top-10/`

Summary

In this section, we have learned how to securely maintain the AWS Lambda service, based on AWS infrastructure – from logging in, to network access, to auditing and monitoring, and security compliance.

Securing Azure Functions

Azure Functions is the Azure function as a service.

It can integrate with other Azure services, such as Azure AD for managing permissions to Azure Functions, Azure Monitor Application Insights for monitoring Azure Functions, and Azure Blob storage for persistent storage.

Best practices for configuring IAM for Azure Functions

Azure AD is the supported service for managing permissions to your Azure Functions.

The best practices are as follows:

- Enable Azure AD authentication for any newly created Azure function by turning on Azure App Service authentication.
- Avoid allowing anonymous access to your Azure function – require clients to authenticate before using Azure Functions.
- Grant minimal permissions for any newly created Azure function using Azure RBAC.
- Prefer to use temporary credentials to your Azure function – use **Shared Access Signature** (**SAS**) tokens to achieve this task.
- Where possible, prefer to use client certificates to authenticate clients to your Azure functions.
- To allow your Azure functions access to Azure resources, use a system-assigned managed identity from Azure AD.
- If you need to store sensitive data (such as credentials), use Azure Key Vault.
- For sensitive environments, encrypt the Azure Functions application settings using customer-managed key management inside Azure Key Vault (as explained in *Chapter 7*, *Applying Encryption in Cloud Services*).

For more information, please refer the following resources:

How to use managed identities for App Service and Azure Functions:

`https://docs.microsoft.com/en-us/azure/app-service/overview-managed-identity?toc=/azure/azure-functions/toc.json`

Azure Functions authorizations:

`https://docs.microsoft.com/en-us/azure/api-management/import-function-app-as-api#authorization`

Use Key Vault references for App Service and Azure Functions:

`https://docs.microsoft.com/en-us/azure/app-service/app-service-key-vault-references`

Best practices for securing data and network access to Azure Functions

Azure Functions can access resources in your Azure subscription – for that reason, it is important to plan before deploying each Azure function.

The best practices are as follows:

- For better protection of your Azure functions, configure the Azure function behind Azure API Gateway.
- Use TLS 1.2 to encrypt sensitive data over the network.
- Create a separate Azure storage account for any newly created Azure function.
- Use Azure network security groups to block outbound traffic from your Azure functions (when internet access is not required).
- Use the Azure VNet service endpoint to control access to your Azure functions.
- Use Azure App Service static IP restrictions to control access to your Azure functions.
- Use either an Azure App Service Standard plan or an Azure App Service Premium plan to configure network isolations of your Azure functions.
- Use Azure Defender for App Service as an extra layer of protection for your Azure functions that have inbound access from the internet.
- Use Azure Web Application Firewall as an extra layer of protection for your Azure functions that have inbound access from the internet.
- Disable and block the use of the FTP protocol with your Azure functions.

For more information, please refer the following resources:

Azure Functions networking options:

```
https://docs.microsoft.com/en-us/azure/azure-functions/
functions-networking-options
```

Secure an HTTP endpoint in production:

```
https://docs.microsoft.com/en-us/azure/azure-functions/
functions-bindings-http-webhook-trigger?tabs=csharp#secure-an-
http-endpoint-in-production
```

IP address restrictions:

```
https://docs.microsoft.com/en-us/azure/azure-functions/
ip-addresses#ip-address-restrictions
```

Azure Functions and FTP:

```
https://docs.microsoft.com/en-us/azure/azure-functions/
functions-deployment-technologies#ftp
```

Protect your web apps and APIs:

```
https://docs.microsoft.com/en-us/azure/security-center/
defender-for-app-service-introduction
```

Best practices for conducting auditing and monitoring in Azure Functions

Auditing is a crucial part of data protection.

As with any other managed service, Azure allows you to enable logging and auditing using the Azure Monitor service.

The best practices are as follows:

- Use the Azure Monitor service to log authentication-related activities of your Azure functions.
- Use the Security Center threat detection capability (Azure Defender).

For more information, please refer the following resources:

Logging and threat detection:

```
https://docs.microsoft.com/en-us/security/benchmark/azure/
baselines/functions-security-baseline#logging-and-threat-
detection
```

Azure security baseline for Azure Functions:

```
https://docs.microsoft.com/en-us/azure/azure-functions/
security-baseline#logging-and-monitoring
```

Best practices for conducting compliance, configuration change, and secure coding in Azure Functions

Serverless, or function as a service, is mainly code running inside a closed, managed environment.

As a customer, you cannot control the underlying infrastructure – as a result, you must invest in secure coding to avoid attackers breaking into your application and causing harm that Azure cannot protect against.

The best practices are as follows:

- Follow the *OWASP Serverless Top 10* project documentation when writing your Azure Functions code.
- Use the Azure security baseline for Azure Functions (Azure Security Benchmark).
- Use the Security Center threat detection capability (Azure Defender).

For more information, please refer the following resources:

Azure security baseline for Azure Functions:

`https://docs.microsoft.com/en-us/security/benchmark/azure/baselines/functions-security-baseline`

Posture and vulnerability management:

`https://docs.microsoft.com/en-us/security/benchmark/azure/baselines/functions-security-baseline#posture-and-vulnerability-management`

OWASP Serverless Top 10:

`https://owasp.org/www-project-serverless-top-10/`

Summary

In this section, we have learned how to securely maintain the Azure Functions service, based on Azure infrastructure – from logging in, to network access, to auditing and monitoring, and security compliance.

Securing Google Cloud Functions

Google Cloud Functions is the GCP function as a service.

It can integrate with other GCP services, such as Google Cloud IAM for managing permissions to Google Cloud Functions, Google Cloud Audit Logs for monitoring Google Cloud Functions, and Google Cloud Storage for persistent storage.

Best practices for configuring IAM for Google Cloud Functions

Google Cloud IAM is the supported service for managing permissions on your Google Cloud Functions.

The best practices are as follows:

- Use Google Cloud IAM to manage permissions to your Google Cloud functions.
- Grant minimal permissions for accessing and managing the Google Cloud IAM service.
- Create a unique service account for each newly created Google Cloud function with minimal permissions using the Google Cloud IAM service.

For more information, please refer the following resources:

Authorizing access via IAM:

`https://cloud.google.com/functions/docs/securing/managing-access-iam`

Authenticating for invocation:

`https://cloud.google.com/functions/docs/securing/authenticating`

Securing access with identity:

`https://cloud.google.com/functions/docs/securing#identity`