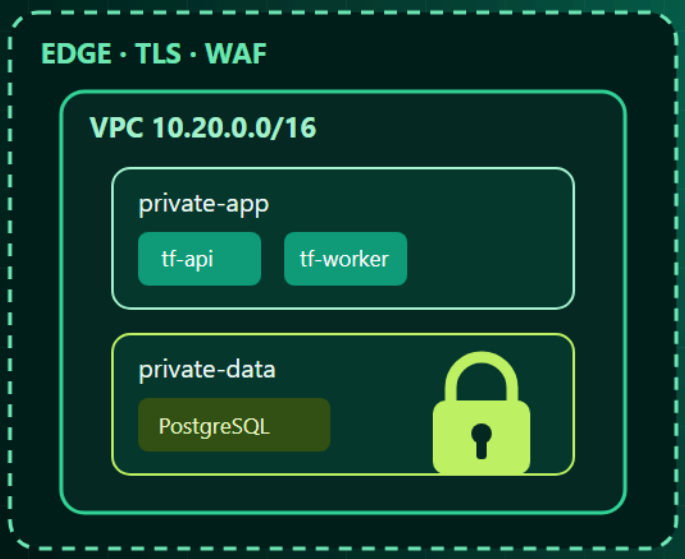
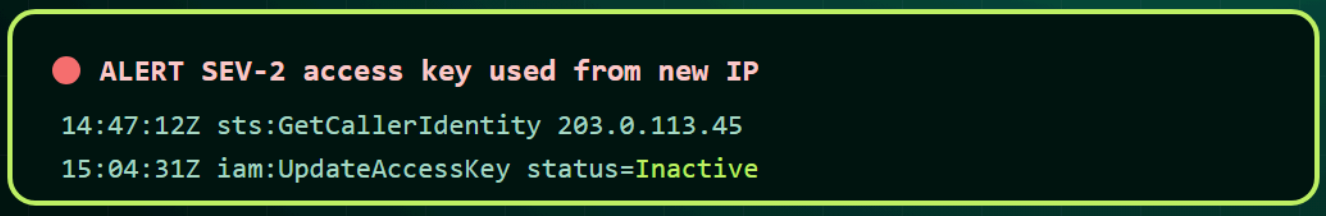






MFA · SSO · OIDC | KMS · Key Vault | CIS · NIST · SOC 2

A COMPLETE, PRACTICAL GUIDE

Cloud Security Fundamentals

Identity, secrets, networks, data protection, monitoring, and compliance on AWS and Azure

Least Privilege · IAM & Entra ID · KMS & Key Vault · VPC & VNet
WAF · CI/CD · CloudTrail · Sentinel · Incident Response · SOC 2

IDENTITY · DATA · NETWORK · DETECTION · GOVERNANCE

THENEXBOOKS

Cloud Security Fundamentals

Identity, Secrets, Networks, Data Protection, Monitoring, and Compliance on AWS and Azure
From one over-privileged account to a monitored, audit-ready cloud — built step by step

First Edition

THENEXBOOKS

Cloud Security Fundamentals
Identity, Secrets, Networks, Data Protection, Monitoring, and Compliance on AWS and Azure

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples, policies, and scripts may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Cloud services change frequently: service names, console layouts, defaults, features, and pricing differ over time and between regions. Always consult the provider's current documentation, test changes in a non-production environment first, and never apply a policy to production that you have not reviewed. This book describes security and compliance frameworks for education only; it is not legal advice, and following it does not by itself make any organization compliant with, or certified against, any standard or regulation.

Amazon Web Services, AWS, Amazon S3, Amazon EC2, AWS Identity and Access Management (IAM), AWS Key Management Service (KMS), Amazon GuardDuty, AWS CloudTrail, AWS Security Hub, and related marks are trademarks of Amazon.com, Inc. or its affiliates. Microsoft, Azure, Microsoft Entra, Microsoft Defender, Microsoft Sentinel, Windows, and PowerShell are trademarks of the Microsoft group of companies. GitHub is a trademark of GitHub, Inc. Terraform is a trademark of HashiCorp, Inc. CIS, CIS Benchmarks, and CIS Controls are trademarks of the Center for Internet Security, Inc. ISO and IEC are trademarks of their respective organizations. SOC 2 is associated with the American Institute of Certified Public Accountants (AICPA). PCI DSS is a standard of the PCI Security Standards Council. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by any of these companies or organizations.

Tallyfern, the startup used as the running example in this book, is fictional, as are its employees, customers, cloud accounts, subscriptions, resource names, IP addresses, incidents, and all sample data. Account IDs, keys, and addresses are documentation examples or reserved ranges; email addresses use reserved example domains. All security content is defensive: it is intended to help you protect systems you own or are authorized to manage.

Preface

Moving to the cloud changes what "security" means. There is no server room to lock and no network cable to unplug. Everything — every virtual machine, database, storage bucket, firewall rule, and permission — is created and changed through an API call. That is what makes the cloud fast, and it is also what makes it fragile: one overly broad policy, one access key pasted into the wrong file, or one storage bucket opened "just for a test" can expose data to the whole internet in seconds.

The good news is that the cloud also gives you better security tools than most organizations ever had on premises: fine-grained identity policies, managed encryption keys, detailed audit logs of every action, organization-wide guardrails, and services that continuously check your configuration. The challenge is knowing which of the hundreds of settings matter, how they fit together, and in what order to tackle them.

This book teaches cloud security from the ground up and shows you how to apply it on both Amazon Web Services and Microsoft Azure. We do not just list services. We follow one small company as it moves its web application to the cloud and grows up — from a single over-privileged account to a structured, monitored, audit-ready environment — and we explain *why* each control exists, so you can make good decisions in your own environment.

Who This Book Is For

- **Developers and DevOps engineers** who build on AWS or Azure and have just been told they are now responsible for securing it.
- **Platform, cloud, and IT engineers** who manage accounts, subscriptions, networks, and identities and want a clear, structured approach.
- **Technical founders and engineering leads** at startups who must answer customer security questionnaires and prepare for audits such as SOC 2.
- **Students and career changers** preparing for a first cloud security role who want a practical, vendor-balanced foundation.

You should know what a virtual machine, a storage bucket, and a region are, and be comfortable running commands in a terminal. No prior security experience is required.

What You Will Learn

- How shared responsibility works, how to think in principles such as least privilege, defense in depth, and Zero Trust, and how to threat model a cloud application.
- How to structure AWS Organizations and Azure management groups, and how to set baselines and guardrails with service control policies and Azure Policy.

- How to design identity and access: IAM and Microsoft Entra ID, federation and single sign-on, least-privilege policies, Azure RBAC, workload identities, OIDC for pipelines, MFA, Conditional Access, break-glass accounts, and access reviews.
- How to protect secrets and data: cryptography essentials, AWS KMS and Azure Key Vault, envelope encryption, secrets management and scanning, storage and database protection, and data classification.
- How to secure networks and workloads: VPCs and VNets, security groups and NSGs, private endpoints, egress control, TLS, WAF and DDoS protection, VMs, containers, serverless, and CI/CD pipelines.
- How to detect and respond: logging with CloudTrail and Azure Monitor, threat detection with GuardDuty, Defender for Cloud, and Microsoft Sentinel, incident response, and a complete incident case study.
- How to manage posture and governance: findings and auto-remediation, CIS, NIST CSF, ISO/IEC 27001, SOC 2, and PCI DSS explained, policies, tagging, exceptions, risk registers, and evidence.

How This Book Is Organized

Part I — Foundations (Chapters 1–5) covers shared responsibility, core principles, threat modeling, account and subscription structure, and secure baselines. **Part II — Identity and Access** (Chapters 6–10) covers identities and federation, least-privilege policies, workload identity, strong authentication, and access governance. **Part III — Secrets and Data Protection** (Chapters 11–15) covers cryptography, key management, secrets, object storage, and databases. **Part IV — Network and Workload Security** (Chapters 16–20) covers networks, traffic control, the edge, compute and containers, and CI/CD. **Part V — Detection and Response** (Chapters 21–24) covers logging, threat detection, incident response, and a full incident case study. **Part VI — Posture, Governance, and Compliance** (Chapters 25–28) covers posture management, compliance frameworks, governance in practice, and a complete security program and roadmap. The appendices provide a cheat sheet, templates and reference material, a production security checklist, and a glossary.

Each chapter ends with "Try It Yourself" exercises and a **Key Takeaways** box for quick review.

The Running Example: Tallyfern

Throughout the book we secure **Tallyfern**, a fictional startup that offers online invoicing and payment links to freelancers and small agencies in the UK and EU. Tallyfern's web app runs on AWS: containers behind a load balancer, a PostgreSQL database, storage buckets for invoice PDFs and receipt images, a job queue, and a CI/CD pipeline. Its staff sign in with Microsoft Entra ID, and a reporting workload, Tallyfern Insights, runs on Azure. Card payments are handled by a third-party payment processor, but Tallyfern stores its customers' bank account details for payouts — so the stakes are real.

When we meet Tallyfern in Chapter 1, everything runs in one AWS account, several engineers share administrator access, secrets live in environment files, and a "temporary" test bucket is public. Over the following chapters, Aisha Rahman, Tallyfern's first security engineer, works with the platform team to restructure accounts, tighten identity, encrypt data, lock down the network, harden the pipeline, turn on logging and detection, handle a real incident, and prepare for a SOC 2 audit. Every account, role, policy, and resource name is consistent from the first chapter to the last.

Versions and Conventions

Examples target **AWS and Microsoft Azure as offered at the time of writing**. Cloud providers rename services, redesign consoles, and change defaults frequently; where something is new, in preview, or recently renamed, a Note says so. If a page or menu is not exactly where the book says, use the console's search box and the provider's current documentation. The book never asks you to deploy anything: you can follow every chapter with a text editor, Python 3, OpenSSL, and a terminal, and apply the ideas in your own sandbox account if you have one. Command output from cloud CLIs is shown as "something like" because IDs and timestamps will differ in your environment.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can cause data exposure, outages, or lockouts.

Console paths are written in bold with arrows, such as **IAM > Roles > Create role**. Policies, commands, and code appear in `monospace`; long lines are wrapped to fit the page. Names follow a consistent convention — `tallyfern-prod`, `tf-prod-api-task-role`, `alias/tf-prod-data`, `kv-tf-insights-prod` — summarized in Appendix B. Python scripts use only the standard library of Python 3.12.

Contents

Preface

PART I — FOUNDATIONS

1 Cloud Security in Context: Shared Responsibility, Threats, and Tallyfern

2 Core Principles: Least Privilege, Defense in Depth, and Zero Trust

3 Threat Modeling a Cloud Application

4 Organizing the Cloud: AWS Organizations, Azure Management Groups, and Landing Zones

5 Secure Baselines and Guardrails: Root Protection, SCPs, and Azure Policy

PART II — IDENTITY AND ACCESS

6 Identity Fundamentals: IAM, Microsoft Entra ID, and Federation

7 Writing Least-Privilege Policies: AWS IAM JSON and Azure RBAC

8 Workload Identity: IAM Roles, Managed Identities, and OIDC Federation

9 Strong Authentication: MFA, Single Sign-On, Conditional Access, and Break-Glass Accounts

10 Governing Access: Reviews, Permission Boundaries, and Privilege Escalation Paths

PART III — SECRETS AND DATA PROTECTION

11 Cryptography Essentials for Cloud Engineers

12 Key Management: AWS KMS, Azure Key Vault, and Envelope Encryption

13 Secrets Management: Storage, Rotation, and Secret Scanning

14 Protecting Object Storage: Amazon S3 and Azure Blob Storage

15 Protecting Databases and Sensitive Data: Encryption, Classification, and Backups

PART IV — NETWORK AND WORKLOAD SECURITY

16 Network Foundations: VPCs, VNets, Subnets, and Segmentation

17 Controlling Traffic: Security Groups, NSGs, Firewalls, and Private Endpoints

18 Securing the Edge: TLS, Load Balancers, WAF, and DDoS Protection

19 Compute and Container Security: VMs, Containers, and Serverless

20 Securing CI/CD and the Software Supply Chain

PART V — DETECTION AND RESPONSE

21 Logging That Matters: CloudTrail, Azure Monitor Logs, and Flow Logs

22 Threat Detection: GuardDuty, Defender for Cloud, Sentinel, and Custom Alerts

23 Incident Response in the Cloud: Preparation, Playbooks, and Evidence

24 Incident Case Study: The Leaked Access Key

PART VI — POSTURE, GOVERNANCE, AND COMPLIANCE

25 Cloud Security Posture Management: Findings, Benchmarks, and Auto-Remediation

26 Compliance Frameworks Explained: CIS, NIST CSF, ISO 27001, SOC 2, and PCI DSS

27 Governance in Practice: Policies, Tagging, Exceptions, Risk, and Evidence

28 Putting It All Together: Tallyfern's Security Program and Roadmap

APPENDICES

A Cloud Security Cheat Sheet

B Templates and Reference

C Production Security Checklist

D Glossary

PART I

Foundations

What makes cloud security different, who is responsible for what, the principles that guide every decision, how to find threats before attackers do, and how to structure accounts and subscriptions with secure baselines and guardrails from day one.

1 · Cloud Security in Context

2 · Core Principles

3 · Threat Modeling a Cloud Application

4 · Organizing the Cloud

5 · Secure Baselines and Guardrails

CHAPTER 1 · PART I — FOUNDATIONS

Cloud Security in Context: Shared Responsibility, Threats, and Tallyfern

Before you write a single policy, you need a clear picture of what is different about securing the cloud, which parts of the job belong to you, and what actually goes wrong. In this chapter we build that picture and meet Tallyfern, the startup whose security journey we follow for the rest of the book.

Why Cloud Security Feels Different

If you have ever secured a server room, you know the classic picture of security: a building with a locked door, a firewall at the edge of the network, and a team that controls every machine plugged into it. The cloud breaks almost every part of that picture. You still have servers, networks, and data, but you no longer touch any of them directly. Everything you do, you do by calling an API.

That single fact changes the shape of the whole discipline. Four differences are worth naming right away, because they come up again in every chapter of this book.

1. Everything is an API call

Creating a virtual machine, opening a firewall port, reading a file from storage, deleting a database backup: in the cloud, each of these is a request to a provider endpoint, signed with a credential. The console you click through is just a friendly client for the same APIs that the command line and the SDKs use. This is wonderful for automation, and it is also the reason an attacker who obtains a credential does not need to break into a building or a network. They can do from a laptop anywhere in the world exactly what your engineers do from theirs.

The upside is that every action is also *recordable*. Because there is no way to change cloud resources except through the API, the provider can log every call: who made it, from where, when, and whether it succeeded. Services such as AWS CloudTrail and the Azure Activity log are the backbone of detection and investigation, and we spend most of Part V on them.

2. Identity is the new perimeter

In a traditional data center, being "inside the network" granted a lot of implicit trust. In the cloud, the most important question is not *where* a request comes from but *who* made it and what that identity is allowed to do. A storage bucket with a correct policy is safe even though it lives on the public internet; a bucket with a careless policy is exposed even if every network rule is perfect. That is why we spend all of Part II on identity and access, and why "stolen credentials" appears at the top of almost every list of cloud threats.

3. Everything is configuration

A cloud environment is, in a very real sense, a large collection of settings. Is this bucket private? Is encryption on? Is this database reachable from the internet? Who can assume this role? Each answer is a value in a configuration document somewhere. That means most cloud security problems are configuration problems, and most cloud security work is about getting configuration right and keeping it right. It also means that configuration can be written as code, reviewed like code, tested like code, and scanned automatically. Infrastructure as code (we use Terraform for AWS and Bicep for Azure in this book) is one of the strongest security tools you have.

4. The speed of change

A team can create, change, and destroy hundreds of resources in a day. Providers release new services and new settings constantly. Default behaviors change: for example, new Amazon S3 buckets have had Block Public Access turned on and ACLs disabled by default since 2023. A security review that was accurate last quarter may be stale today. Effective cloud security therefore relies on **continuous** controls (guardrails that apply automatically, detections that run all the time, posture checks that re-evaluate every resource) rather than one-off audits.

> **NOTE**
>
> Cloud services change constantly. Consoles get redesigned, services get renamed, and defaults change. Throughout this book we say "at the time of writing" or "in recent versions" for anything that is not a long-stable fundamental. Always check the provider's current documentation before you rely on a specific default, limit, or menu label.

The Shared Responsibility Model

The most important idea in cloud security fits in one sentence: **the provider is responsible for the security *of* the cloud, and you are responsible for security *in* the cloud.** Both AWS and Microsoft publish a version of this model, and although the diagrams look different, the substance is the same.

"Security of the cloud" means the things you cannot see or touch: the physical data centers, the hardware, the hypervisors that isolate your virtual machines from other customers, the global network, and the internal workings of managed services. You could not secure those even if you wanted to, and you have to trust the provider (and the provider's independent audit reports, which we discuss in Chapter 26) to do it.

"Security in the cloud" means everything you put there and every decision you make about it: the identities you create and the permissions you grant, the data you store and whether you encrypt it, the network rules you write, the code you deploy, the operating systems you patch on your own virtual machines, and the configuration of every service you turn on.

The line moves with the service model

Where exactly the line falls depends on how much of the stack the provider runs for you. The three classic service models are:

- **Infrastructure as a Service (IaaS)**: you rent virtual machines, disks, and networks. The provider secures the hardware and the hypervisor; you secure the operating system, its patches, the software on it, the network rules, the identities, and the data. Amazon EC2 and Azure Virtual Machines are typical examples.
- **Platform as a Service (PaaS)**: you hand the provider your code or your data and it runs the platform. You no longer patch the operating system or the database engine, but you still configure access, network exposure, encryption choices, and everything about your own code. Amazon RDS, AWS Lambda, Azure Functions, and Azure SQL Database sit here.
- **Software as a Service (SaaS)**: you use a finished application. The provider runs almost everything; you are still responsible for who has accounts, how they sign in, what data you put in, and how you configure sharing. Microsoft Entra ID and GitHub are SaaS from Tallyfern's point of view.

Notice what never moves to the provider, whichever model you pick: **identities, data, and configuration**. You always decide who can do what, what data goes where, and how the service is set up. That is why those three topics dominate this book.

Shared responsibility for Tallyfern's services

A useful exercise for any team is to write the model out for the specific services they use. Here is the version that Tallyfern's security engineer produces in her first week (we meet her shortly):

Service	Model	Provider secures	Tallyfern secures
ECS on Fargate (`tf-api`, `tf-web`, `tf-worker`)	Container platform	Hosts, kernel, container runtime isolation	Images, task roles, secrets, network rules, app code
Amazon RDS for PostgreSQL	PaaS	OS and engine patching mechanics, storage hardware	Network exposure, encryption choice, DB users, backups policy
Amazon S3	PaaS	Durability, storage infrastructure	Bucket policies, public access settings, encryption keys, data
AWS Lambda (`tf-prod-thumbnailer`)	Serverless	Runtime hosts, scaling, isolation	Function code, dependencies, execution role, triggers
Azure Functions (`func-tf-insights-prod`)	Serverless	Hosting platform	Code, managed identity permissions, networking, app settings
Azure SQL Database	PaaS	Engine, patching, platform encryption	Authentication mode, firewall, auditing, data

Microsoft Entra ID	SaaS	Directory service operation	Users, groups, MFA, Conditional Access, app consent
GitHub (code and CI)	SaaS	Platform, hosted runners	Repository access, workflow permissions, secrets, branch rules

> **WARNING**
> "It's a managed service, so it's secure" is one of the most expensive misunderstandings in cloud computing. A managed database can be perfectly patched and still be reachable from the entire internet with a weak password. The provider manages the service; you manage how it is exposed and who can use it.

What Actually Goes Wrong: Common Threat Categories

Security work only pays off when it is aimed at real risks. A recurring theme in published incident reports is that most cloud incidents involve customer-side misconfiguration and stolen credentials, not exotic attacks on the provider's infrastructure. We will not quote percentages or costs here (they vary widely between reports, and many are not comparable), but the pattern is consistent enough to shape our priorities. Here are the six threat categories we return to throughout the book.

Stolen or leaked credentials

A password phished from an engineer, an access key committed to a public repository, a token copied out of a CI log, a session cookie stolen by malware: any of these gives an attacker the same power as the legitimate owner. Long-lived credentials are the most dangerous kind because they keep working until someone notices. AWS IAM user access keys, which start with `AKIA`, never expire on their own. Temporary credentials from roles, which start with `ASIA` and come with a session token, expire automatically. A large part of Part II is about replacing the first kind with the second.

Misconfiguration

A storage bucket readable by anyone, a database with a public endpoint, a security group open to `0.0.0.0/0` on SSH, logging turned off, encryption never enabled, a role that trusts too broadly. Misconfigurations are usually honest mistakes made under time pressure, and they are made worse by the speed of change: a test resource created "just for an afternoon" can live for years. Guardrails (Chapter 5), posture management (Chapter 25), and infrastructure as code reviews (Chapter 20) are the main defenses.

Exposed secrets

Closely related to stolen credentials, but worth its own category: database passwords, third-party API keys, signing secrets, and private keys that end up in source code, container images, environment files, tickets, chat messages, or logs. Chapter 13 covers where secrets leak and how to find them before an

attacker does.

Vulnerable workloads

Your application code, its dependencies, the operating systems on your virtual machines, and the base layers of your container images all contain bugs, and some of those bugs are security vulnerabilities. A web application flaw can let an attacker run code inside your container, and from there use whatever cloud credentials that container holds. This is why workload permissions must be tight (Chapter 8) and why compute hardening matters (Chapter 19).

Supply chain compromise

Modern software is assembled from hundreds of open-source packages, base images, CI actions, and infrastructure modules written by other people. If any of them is compromised, or if the pipeline that builds and deploys your code can be tampered with, the attacker's code runs with your deploy permissions. Chapter 20 is dedicated to CI/CD and supply chain security.

Insider misuse and mistakes

Not every threat comes from outside. A disgruntled employee, a contractor whose access was never removed, or simply a tired engineer running a command against production instead of staging: insiders already hold valid credentials. Separation of duties, access reviews, just-in-time privileges, and good audit logs (Chapters 2, 10, and 21) are how we reduce both malicious and accidental insider risk.

> **TIP**
>
> When you assess any cloud design, ask the same three questions every time: "What happens if this credential leaks?", "What happens if this setting is wrong?", and "Would we notice?" Those three questions alone catch a surprising share of real problems.

Meet Tallyfern

To keep every chapter concrete, we follow one fictional company, with names and IDs that stay consistent throughout the book.

Tallyfern Ltd offers online invoicing and payment links for freelancers and small agencies in the UK and the EU. A freelance designer creates an invoice in the web app, sends the customer a payment link, and gets paid; Tallyfern then pays out to the freelancer's bank account. The company has 14 employees. Its marketing site lives at `tallyfern.example`, the web app at `app.tallyfern.example`, and the public API at `api.tallyfern.example`.

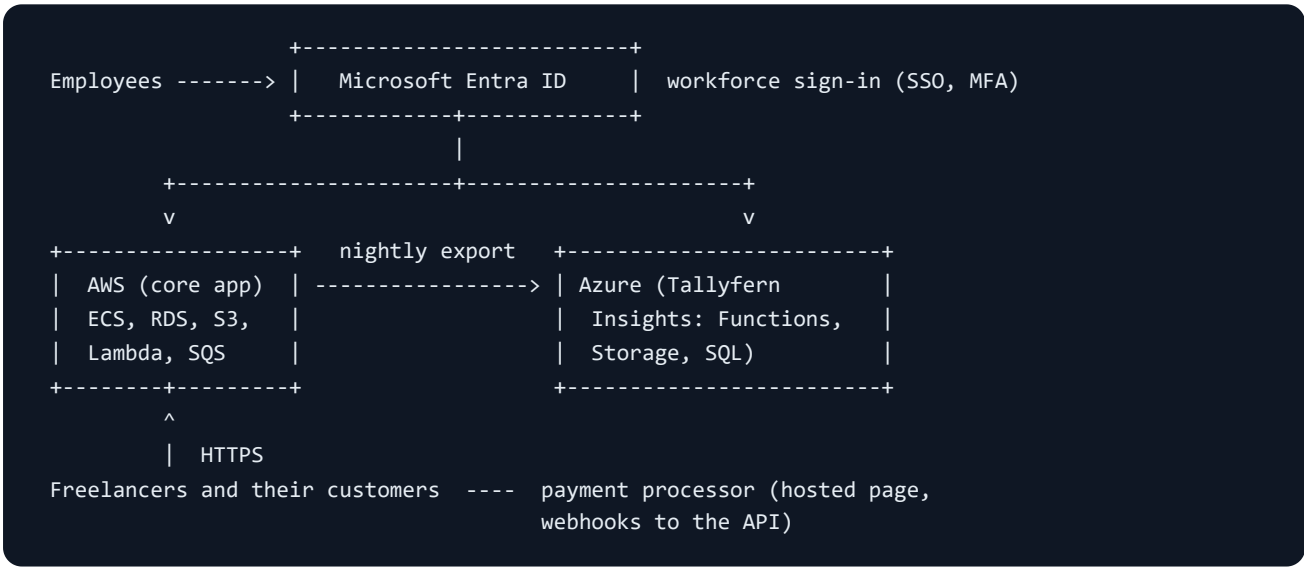
Tallyfern calls itself "fintech-lite", and the "lite" part is a deliberate design decision. Card payments are taken on the **payment processor's hosted payment page** (a third-party processor we always call simply "the payment processor", with API host `api.processor.example`). Tallyfern never sees or stores

card numbers; it stores only the processor's payment token and the card brand and last four digits. That decision keeps a lot of regulatory scope out of Tallyfern's systems, which we explore in Chapter 26. However, Tallyfern *does* store its customers' bank account details (IBANs) for payouts. Those IBANs are the company's most sensitive data, and protecting them is a thread that runs through Parts III and IV.

The architecture in brief

The core application runs on AWS in the `eu-west-1` Region. Three containerized services run on Amazon ECS with AWS Fargate: `tf-api` (the API, on port 8080), `tf-web` (the web front end), and `tf-worker` (which renders invoice PDFs, sends outbound webhooks, and runs a nightly export). Data lives in an Amazon RDS for PostgreSQL database and in S3 buckets for invoice PDFs and receipt images. A small Lambda function resizes uploaded receipt images.

Employees sign in with **Microsoft Entra ID** (formerly Azure Active Directory), because the company already uses Microsoft 365 for email. A reporting product called **Tallyfern Insights**, which produces monthly financial reports for business customers, runs on **Microsoft Azure** in the West Europe Region. Code lives on GitHub and is deployed with GitHub Actions.



The starting state

Tallyfern grew fast, and like many startups it took shortcuts. When our story begins, the situation looks like this:

- **One AWS account for everything.** Production, staging experiments, developer sandboxes, CI tooling, and logs all live in a single account (the account that later becomes `tallyfern-prod`). Anyone with broad access to that account can touch production.
- **Shared, over-privileged IAM users.** An IAM user called `admin-shared` with the `AdministratorAccess` policy is used by several people. Individual engineers also have IAM users with long-lived access keys, such as `mateo-cli`, and a contractor's CI job uses one called `kenji-ci`.

- **A public test bucket.** A bucket called `tallyfern-test-public` was made publicly readable months ago so that a partner could download test files.
- **Secrets in environment files.** The API reads its database password and the processor API key from a `.env` file that is copied around between laptops and baked into deployment scripts.
- **Thin logging.** Nobody set up a CloudTrail trail, so the only record of API activity is the 90-day Event history that CloudTrail keeps by default for management events.
- **Azure set up by hand.** Tallyfern Insights runs in a single subscription where three people hold the Owner role "to keep things simple".

None of this is unusual. Each shortcut made sense to someone at the time. The problem is that together they create a situation where one leaked credential, one careless click, or one compromised laptop could expose customer data, and nobody would be able to tell afterwards exactly what happened.

The people

We will meet several Tallyfern employees along the way. The main ones are:

Person	Role	In this book
Nadia Okoro	CTO, executive security owner	Sponsors the program, approves risk acceptances
Aisha Rahman	First dedicated security engineer	Drives the program; you follow her work
Sam Whitfield	Lead platform engineer	Owns the AWS Organization, Terraform, Azure subscriptions
Mateo Ruiz	Backend developer	Owns a legacy IAM user that matters later
Jonah Abara	SRE, on-call	Incident commander when things go wrong
Grace Lindqvist	Head of finance and compliance	Owns the SOC 2 project
Kenji Mori	Front-end contractor	Example for offboarding and access reviews

The story begins when Aisha Rahman joins as Tallyfern's first dedicated security engineer. A large business customer has sent a security questionnaire, the board has asked whether the company is "secure", and Nadia knows that the honest answer is "we don't really know". Aisha's job is to find out, and then to fix what matters most first.

Aisha's First Week: Taking Inventory

You cannot secure what you do not know about. Aisha's first week is not spent writing policies; it is spent looking. She talks to every engineer, reads the deployment scripts, and walks through the AWS console and Azure portal with Sam, writing down what she finds. Her questions are simple:

1. Which cloud accounts and subscriptions exist, and who owns each one?
2. Which identities exist (human users, shared users, service users, keys), and what can each do?
3. Where does customer data live, and how sensitive is it?
4. What is reachable from the internet?
5. Where are the secrets, and who can read them?
6. What is logged, where, and for how long?

In her first days, Aisha uses a handful of read-only commands. The first one every cloud engineer should know answers the question "who am I right now?":

Terminal (AWS CLI)

```
aws sts get-caller-identity
```

You should see something like this (IDs will differ):

Output (something like)

```
{
    "UserId": "AIDAEXAMPLEUSERID1234",
    "Account": "444455556666",
    "Arn": "arn:aws:iam::444455556666:user/admin-shared"
}
```

That output alone tells a story: Aisha was handed the shared admin user to do her review. Every action she takes will be logged as `admin-shared`, indistinguishable from the actions of the other people who use it. That is a finding in itself.

On the Azure side, the equivalent first command shows which tenant and subscription the CLI is currently pointed at:

Azure CLI

```
az account show --query "{name:name, id:id, tenant:tenantId}"
```

Output (something like)

```
{
  "id": "aaaaaaaa-1111-2222-3333-000000000001",
  "name": "sub-tf-insights-prod",
  "tenant": "11111111-2222-3333-4444-555555555555"
}
```

She records each observation in a simple spreadsheet with four columns: area, item, observation, and a rough severity. Rough is fine at this stage. The goal is to make the invisible visible so the team can agree on priorities.

Summarizing the inventory with a script

Aisha exports her spreadsheet as CSV and writes a tiny Python script to summarize it. It uses only the standard library, so it runs anywhere Python 3.12 is installed. Here is the CSV she ends the week with:

first-week-inventory.csv

```
area,item,observation,severity
identity,admin-shared,IAM user with AdministratorAccess; password shared by 4 people,HIGH
identity,mateo-cli,IAM user with long-lived access key (AKIA...) on a laptop,HIGH
identity,kenji-ci,IAM user with access key used by a contractor's CI job,HIGH
identity,root user,MFA enabled; no access keys; email is a personal inbox,MEDIUM
storage,tallyfern-test-public,bucket policy allows public read; holds test files,HIGH
storage,tallyfern-prod-invoices,private; default SSE-S3 encryption; versioning off,MEDIUM
secrets,api .env file,database password and processor API key in plain text,HIGH
secrets,CI variables,AWS access key stored as a CI secret,MEDIUM
network,production database,"PostgreSQL in default VPC, open to the internet",HIGH
logging,CloudTrail,only the 90-day Event history; no trail to S3,MEDIUM
logging,app logs,no retention setting; some lines include full request bodies,MEDIUM
azure,sub-tf-insights-prod,single subscription; three people hold Owner,MEDIUM
```

Notice the double quotes around the network observation. That text contains a comma, and in CSV an unquoted comma would split the field in two and shift the severity into the wrong column. Quoting the field is how CSV escapes embedded commas, and Python's `csv` module handles it for you. Here is the script:

inventory_summary.py

```
"""Summarize a first-week security inventory CSV (stdlib only)."""
import csv
import sys
from collections import Counter

ORDER = ["HIGH", "MEDIUM", "LOW"]


def load(path):
    with open(path, newline="", encoding="utf-8") as f:
        return list(csv.DictReader(f))


def main(path):
    rows = load(path)
    by_sev = Counter(r["severity"] for r in rows)
    by_area = Counter(r["area"] for r in rows)
    print(f"{len(rows)} observations")
    print("By severity: " + ", ".join(f"{s} {by_sev[s]}" for s in ORDER if by_sev[s]))
    print("By area:     " + ", ".join(f"{a} {n}" for a, n in by_area.most_common()))
    print("\nHIGH items to fix first:")
    for r in rows:
        if r["severity"] == "HIGH":
            print(f"  [{r['area']}] {r['item']}: {r['observation']}")
```

```
if __name__ == "__main__":
    main(sys.argv[1] if len(sys.argv) > 1 else "first-week-inventory.csv")
```

PowerShell

```
py -3.12 inventory_summary.py first-week-inventory.csv
```

Output

```
12 observations
By severity: HIGH 6, MEDIUM 6
By area:     identity 4, storage 2, secrets 2, logging 2, network 1, azure 1

HIGH items to fix first:
  [identity] admin-shared: IAM user with AdministratorAccess; password shared by 4 people
  [identity] mateo-cli: IAM user with long-lived access key (AKIA...) on a laptop
  [identity] kenji-ci: IAM user with access key used by a contractor's CI job
  [storage] tallyfern-test-public: bucket policy allows public read; holds test files
  [secrets] api .env file: database password and processor API key in plain text
  [network] production database: PostgreSQL in default VPC, open to the internet
```

The script is deliberately small, but the habit behind it matters: **keep security findings as data**, not as prose buried in a document. Data can be sorted, counted, compared week to week, and fed into other tools. We build on this habit in Chapter 25, where findings from cloud posture tools are triaged by a script in the same spirit.

Agreeing on priorities

At the end of the week, Aisha presents the summary to Nadia and Sam. She does not ask for everything at once. Instead, she proposes a short list of "stop the bleeding" actions for the next two weeks and a longer program for the following months:

- **Now:** remove public access from `tallyfern-test-public` (after checking with the partner), move the production database off the internet, turn on a CloudTrail trail to S3, and stop sharing the `admin-shared` user.
- **Next:** split the single account into a multi-account AWS Organization, put workforce sign-in behind Entra ID and AWS IAM Identity Center, and move secrets into a secrets manager.
- **Later:** encryption key management, network segmentation, CI/CD hardening, detection and incident response, and the compliance work that business customers are asking about.

Nadia approves, with one condition that turns out to be important: every change must be made through Terraform or Bicep and reviewed, not clicked in the console. Security fixes made by hand drift back over time; fixes made as code stay fixed.