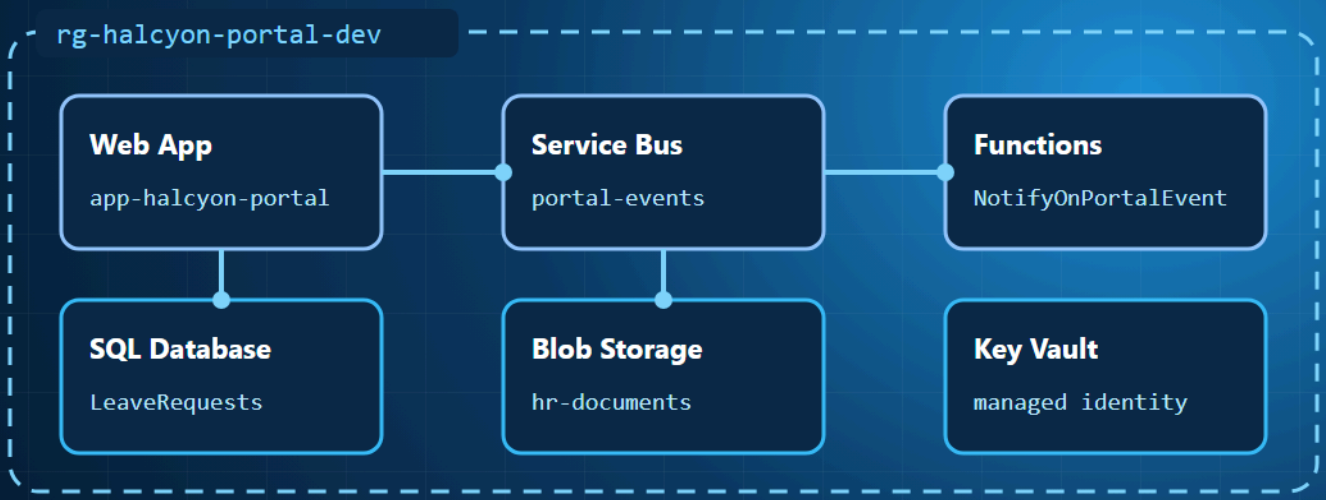



```
resource web 'Microsoft.Web/sites@2023-12-01' = {
  name: 'app-halcyon-portal-dev'
  location: location
  identity: { type: 'UserAssigned' ... }
  properties: { httpsOnly: true }
}
```

Leave request
LR-2026-0008

Employee	HL-1004
Dates	Nov 23 – 27
Working days	4

Approve | Reject

```
$ az deployment group create -g rg-halcyon-portal-dev ...
```

BUILD · DEPLOY · RUN

Microsoft Azure for Developers

Build, deploy, and run cloud applications with C# and .NET

App Service · Functions · Container Apps · SQL · Cosmos DB
Service Bus · Key Vault · Bicep · GitHub Actions · Monitor

IDENTITY · COMPUTE · DATA · INTEGRATION · DEVOPS · COST

THENEXBOOKS

Microsoft Azure for Developers

Build, Deploy, and Run Cloud Applications
From your first resource group to a secure, monitored, production-ready employee portal — built step by step with C# and .NET

First Edition

THENEXBOOKS

Microsoft Azure for Developers
Build, Deploy, and Run Cloud Applications

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, Bicep templates, scripts, and workflow files 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 or charges resulting from the use of the information contained herein. Microsoft Azure changes constantly: services, tiers, portal screens, command-line options, default settings, limits, regional availability, and prices change over time, and features in preview may change or be withdrawn. Always check the current Microsoft documentation and the Azure Pricing Calculator before you rely on a detail, and delete the resources you create while learning so that they do not continue to incur charges.

Microsoft, Azure, Microsoft Entra, Microsoft 365, Visual Studio, Visual Studio Code, .NET, C#, SQL Server, PowerShell, and Windows are trademarks of the Microsoft group of companies. GitHub and GitHub Actions are trademarks of GitHub, Inc. Kubernetes is a trademark of The Linux Foundation. Docker is a trademark of Docker, Inc. Amazon Web Services and AWS are trademarks of Amazon.com, Inc. or its affiliates. Python is a trademark of the Python Software Foundation. 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 Microsoft Corporation.

Halcyon Logistics, the Halcyon Portal, its employees, requests, documents, partners (including "PeopleLedger" and "Paylane"), and all sample data used as the running example in this book are fictional. Any resemblance to real companies or people is coincidental. Domain names use the reserved `example.com` domain, and identifiers such as tenant and subscription IDs are placeholders.

Preface

Most developers meet Azure the same way: a manager says "we're moving to the cloud," a portal login arrives, and suddenly there are hundreds of services, several ways to do everything, and a vocabulary of tenants, subscriptions, resource groups, planes, identities, and SKUs. Tutorials show how to click one service into existence, but rarely how the pieces fit together into an application that is secure, observable, affordable, and deployable by a pipeline instead of by hand.

This book is the guide we wish we had. It teaches Azure from a developer's point of view: what each service is for, how to call it from code, how to describe it as infrastructure as code, how to connect services without passwords, and how to run the result in production. We do not tour every product. We build one realistic application from an empty subscription to a production-ready system, introducing each service at the moment it solves a problem we actually have, and explaining *why* it works the way it does so that you can design your own solutions with confidence.

Azure moves quickly. Throughout the book we say "at the time of writing" when a detail is likely to change, we label preview features as preview, and we never quote prices: costs depend on your region, your agreement, and choices you make, so we teach you what drives cost and how to estimate it yourself with the Azure Pricing Calculator.

Who This Book Is For

- **Application developers new to Azure** who can already program and understand HTTP, REST, and basic SQL, and now need to build and ship on Azure.
- **.NET and C# developers** who want an idiomatic path: the Azure SDK for .NET, ASP.NET Core, Entity Framework Core, Azure Functions, and GitHub Actions.
- **Python and JavaScript developers** who want the concepts and architecture; the main code is C#, and short Python and Node.js equivalents appear where they help.
- **Developers who know another cloud**, such as AWS, and want to map what they know to Azure's model of identity, resources, and services.
- **Team leads and platform engineers** who need a shared, practical baseline for naming, security, deployment, monitoring, and cost.

What You Will Learn

- How Azure is organized — tenants, subscriptions, resource groups, regions, tags — and how Microsoft Entra ID (formerly Azure Active Directory) and Azure RBAC control who can do what.
- How to work from the command line with the Azure CLI, from code with the Azure SDK for .NET, and declaratively with Bicep.
- How to host code on App Service, Azure Functions, and Azure Container Apps, and when AKS or virtual machines are the better choice.

- How to store data in Blob, Queue, and Table Storage, Azure SQL Database with EF Core, and Azure Cosmos DB — and how to back it up and protect it.
- How to connect components with Service Bus and Event Grid, keep secrets in Key Vault, centralize settings and feature flags in App Configuration, and remove passwords entirely with managed identities.
- How to observe the system with Azure Monitor and Application Insights, deploy it with GitHub Actions using OIDC federated credentials, publish APIs with API Management, and lock it down with private networking.
- How to secure, budget, and harden a workload using Defender for Cloud, Azure Policy, Cost Management, and the Azure Well-Architected Framework — finishing with a production go-live case study.

How This Book Is Organized

Part I — Azure Foundations (Chapters 1–5) introduces cloud concepts, Azure's resource hierarchy, identity and access, the Azure CLI and SDK, and Bicep. **Part II — Compute and Hosting** (Chapters 6–10) deploys the web app to App Service, adds Azure Functions and a containerized worker on Azure Container Apps, and compares these with AKS and virtual machines. **Part III — Data and Storage** (Chapters 11–15) covers Blob, Queue, and Table Storage, Azure SQL Database, Cosmos DB, and data protection strategy. **Part IV — Integration and Security** (Chapters 16–20) covers Service Bus, Event Grid, Key Vault, managed identities, and App Configuration. **Part V — Operations and Delivery** (Chapters 21–24) covers monitoring, CI/CD with GitHub Actions, API Management, and networking. **Part VI — Production Readiness** (Chapters 25–28) covers security posture, cost management, reliability and the Well-Architected Framework, and a complete go-live case study. The appendices provide a CLI and Bicep cheat sheet, templates and a service reference, a production checklist, and a glossary.

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

The Running Example: Halcyon Logistics

Throughout the book we build the **Halcyon Portal**, the internal employee portal of **Halcyon Logistics**, a fictional freight and warehousing company. Employees sign in with their work accounts to check leave balances, request time off, raise HR requests such as address changes and employment letters, and upload supporting documents. Managers approve or reject requests, HR administrators handle cases, and everyone receives notifications when something changes. Behind that simple experience sit an ASP.NET Core web app on App Service, Azure Functions for notifications, reminders, auditing, and payroll exports, a containerized document worker on Azure Container Apps, Azure SQL Database, Cosmos DB, Blob, Queue, and Table Storage, Service Bus and Event Grid, Key Vault, App Configuration, Application Insights, API Management for a payroll partner, and a GitHub

Actions pipeline — all described in Bicep and connected with managed identities instead of passwords. The sample data is small enough to check by hand: eight employees, a handful of leave and HR requests, and three documents, with every business rule spelled out so you can verify each result.

What You Need

To follow along you need an Azure subscription in which you can create resource groups (a free account or a pay-as-you-go subscription works — check the current offer terms), the Azure CLI or Azure Cloud Shell, the .NET SDK (the book uses .NET 10), an editor such as Visual Studio Code or Visual Studio, Git, and a GitHub account for the CI/CD chapters. Docker is optional; Chapter 9 shows how to build container images in the cloud instead. Every chapter that creates resources tells you how to clean them up.

Versions and Conventions

Examples target Azure as it is **at the time of writing (2026)**, .NET 10 with C#, the Azure SDK for .NET, a recent Azure CLI, and recent Bicep. Azure portal screens, menu names, command defaults, tiers, and regional availability change frequently; if something is not exactly where the book says, search the portal or run the command with `--help`. Features in preview are labeled as preview. The book never states prices — use the Azure Pricing Calculator and Cost Management for current figures in your region and currency.

> **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 security problems, data loss, outages, or unexpected charges.

Portal paths are written in bold with arrows, such as **Settings > Environment variables**. Commands, code, and resource names appear in `monospace`. Command-line examples use Bash line continuation (`\`), which works in Azure Cloud Shell, macOS, Linux, WSL, and Git Bash; in PowerShell use a backtick instead. Long lines are wrapped to fit the page. Resource names follow one naming convention — `rg-halcyon-portal-dev`, `app-halcyon-portal-dev`, `kv-halcyon-portal-dev` — explained in Chapter 2 and summarized in Appendix B. Some names must be globally unique; if one is taken, add a short suffix and use it consistently.

Contents

Preface

PART I — AZURE FOUNDATIONS

1 Azure for Developers: Cloud Concepts and the Halcyon Portal Plan

2 Organizing Azure: Tenants, Subscriptions, Resource Groups, Regions, and Tags

3 Identity with Microsoft Entra ID and Azure RBAC

4 Working from the Command Line and Code: The Azure CLI and the Azure SDK for .NET

5 Infrastructure as Code with Bicep

PART II — COMPUTE AND HOSTING

6 Azure App Service: Deploying the Halcyon Portal Web App

7 App Service in Production: Slots, Scaling, Settings, and Sign-In

8 Azure Functions: Event-Driven Code with Triggers and Bindings

9 Azure Container Apps: Running the Document Worker

10 Choosing Compute: AKS, Virtual Machines, and a Decision Guide

PART III — DATA AND STORAGE

11 Azure Blob Storage: Secure Document Uploads

12 Queue Storage and Table Storage: Simple Messaging and the Audit Log

13 Azure SQL Database with Entity Framework Core

14 Azure Cosmos DB for NoSQL: The Notification Feed

15 Data Strategy: Choosing Stores, Backups, Redundancy, and Migrations

PART IV — INTEGRATION AND SECURITY

16 Messaging with Azure Service Bus: Queues, Topics, and Subscriptions

17 Event-Driven Integration with Azure Event Grid

18 Secrets, Keys, and Certificates with Azure Key Vault

19 Managed Identities: Passwordless Connections Everywhere

20 Configuration and Feature Flags with Azure App Configuration

PART V — OPERATIONS AND DELIVERY

21 Observability with Azure Monitor and Application Insights

22 CI/CD with GitHub Actions and OIDC Federated Credentials

23 Publishing APIs with Azure API Management

24 Networking Basics: Virtual Networks, Private Endpoints, and Secure Ingress

PART VI — PRODUCTION READINESS

25 Securing Your Azure Workloads: Defender for Cloud, Azure Policy, and Least Privilege

26 Cost Management: Budgets, the Pricing Calculator, and Cost-Aware Design

27 Reliability and the Azure Well-Architected Framework

28 Case Study: Taking the Halcyon Portal to Production

APPENDICES

A Azure CLI and Bicep Cheat Sheet

B Templates and Service Reference

C Production Checklist

D Glossary

PART I

Azure Foundations

What the cloud changes for a developer, how Azure organizes tenants, subscriptions, resource groups, and regions, how Microsoft Entra ID and Azure RBAC decide who can do what, and the three ways you will work with Azure every day: the CLI, the SDK, and Bicep.

1 · Cloud Concepts and the Halcyon Portal Plan

2 · Organizing Azure

3 · Microsoft Entra ID and Azure RBAC

4 · The Azure CLI and the Azure SDK for .NET

5 · Infrastructure as Code with Bicep

CHAPTER 1 · PART I — AZURE FOUNDATIONS

Azure for Developers: Cloud Concepts and the Halcyon Portal Plan

Before we create a single resource, we need a shared map: what the cloud actually changes for a developer, how Azure is organized, who is responsible for what, and what we are going to build together. This chapter gives you that map and introduces the Halcyon Portal, the application that runs through every chapter of this book.

Why a Developer Should Care About the Cloud

If you have spent your career writing code that someone else deploys, the word "cloud" may sound like an operations topic. It is not. On Azure, the decisions that shape your application's cost, reliability, and security are increasingly made in code: in the way you connect to a database, in the identity your app uses to read a file, in the template that describes your infrastructure, and in the pipeline that ships it. Developers who understand the platform write simpler, safer, and cheaper systems than developers who treat it as a remote server room.

Three ideas explain most of what the cloud gives you.

- **On-demand resources.** You ask for a database, a web host, or a message broker with an API call, and it exists a few minutes later. You no longer wait weeks for hardware, and you can create a complete copy of your environment for a test, then delete it the same afternoon.
- **Managed services.** Instead of installing and patching SQL Server on a virtual machine, you create an Azure SQL Database and let Microsoft handle the operating system, the database engine updates, backups, and high availability. Your job moves up the stack: schema, queries, access control, and data design.
- **Pay for what you use.** Most Azure services are billed by consumption: how many instances run, for how long, how much data you store, how many operations you perform, how much data leaves a region. That makes cost a design property: a virtual machine that sits idle all weekend still accrues compute charges, while a consumption-based function that only runs when a message arrives does not run, or bill for executions, while it waits.

This book never quotes prices. Azure prices vary by region, currency, agreement type, and date, and they change. Instead, whenever a decision affects cost, we explain *what drives* the bill so you can model it yourself with the **Azure Pricing Calculator** (`https://azure.microsoft.com/pricing/calculator/`) and watch it in **Cost Management**. Chapter 26 covers this in depth.

The trade-off is a very large number of choices. The goal of this book is not to tour every service but to build one realistic application end to end, explain each choice, and show the alternatives where a different project might choose differently.

Service Models: IaaS, PaaS, Serverless, and SaaS

Cloud services are usually grouped by how much of the stack the provider manages for you. The categories are fuzzy at the edges, but they are a useful vocabulary when you compare options.

Model	You manage	Azure manages	Azure examples
IaaS (Infrastructure as a Service)	OS, patches, runtime, app, data	Hardware, network, virtualization	Virtual Machines, VM Scale Sets, managed disks
PaaS (Platform as a Service)	App code, configuration, data	OS, runtime, scaling, patching	App Service, Azure SQL Database, Container Apps
Serverless	Functions or containers, triggers	Everything else, including scale to zero	Azure Functions, Logic Apps, Container Apps scale to zero
SaaS (Software as a Service)	Users and settings	The entire application	Microsoft 365, GitHub, Dynamics 365

For a developer, the important question is not which box a service lives in but *what you give up and what you get*. A virtual machine gives you full control: any runtime, any agent, any kernel setting. It also gives you every patch, every disk, and every 2 a.m. reboot. App Service gives you a managed host for web apps and APIs: you push code or a container and set a few switches, but you accept its supported runtimes and its sandbox. Azure Functions goes one step further: you write small functions bound to triggers and let the platform decide how many instances to run. Container Apps sits between App Service and Kubernetes: you bring any container, and the platform handles orchestration and event-driven scaling.

The Halcyon Portal uses PaaS and serverless services almost exclusively. That is a deliberate choice for a small internal-application team, and Chapter 10 revisits it with a decision guide that includes Azure Kubernetes Service (AKS) and virtual machines.

The Shared Responsibility Model

Moving to the cloud does not move all responsibility to the cloud provider. Microsoft is responsible for the security *of* the cloud: physical datacenters, hardware, the hypervisor, the global network, and the managed parts of each service. You are responsible for security *in* the cloud: your data, your identities, your access policies, your application code, and the configuration choices you make.

The dividing line moves with the service model. On a virtual machine, you patch the operating system. On App Service, Microsoft patches the operating system and the language runtime images, but you choose which runtime version your app targets and when you upgrade. On Azure SQL Database,

Microsoft patches the engine, but you decide who can connect, whether the server accepts traffic from the internet, and which columns contain sensitive data.

Some responsibilities never leave you, whatever the model:

- **Identity and access.** Who can sign in, which roles they hold, and which applications can call which services.
- **Data.** Classifying it, deciding where it lives, who can read it, how long you keep it, and how you delete it.
- **Your code.** Input validation, authorization checks on every request, dependency updates, and secret handling.
- **Configuration.** A storage account that allows anonymous public access is a configuration choice, not a platform flaw.

> **WARNING**
> Most cloud security incidents trace back to configuration and identity mistakes on the customer side: a secret committed to Git, a storage container left public, an overly broad role assignment, or a firewall rule that admits the whole internet. The platform cannot protect you from settings you chose. This book repeatedly takes the safer default even when it costs a few extra steps.

Control Plane and Data Plane

One distinction will save you hours of confusion later: every Azure service has a **control plane** and a **data plane**.

- The **control plane** is how you *manage* resources: create a storage account, change its redundancy, delete a database, scale a web app. All control-plane operations go through **Azure Resource Manager (ARM)**, a single API at `https://management.azure.com`. The portal, the Azure CLI, Azure PowerShell, Bicep templates, Terraform, and the management SDKs are all clients of ARM.
- The **data plane** is how you *use* a resource: upload a blob, run a SQL query, send a Service Bus message, read a Key Vault secret. Each service exposes its own data-plane endpoint, such as `https://sthalcyonportaldev.blob.core.windows.net/` for blob storage or `sbns-halcyon-portal-dev.servicebus.windows.net` for Service Bus.

The two planes are secured separately. Having permission to *manage* a storage account does not automatically give you permission to *read the files* in it. Chapter 3 explains why a developer with the Contributor role on a resource group still gets an authorization error when uploading or listing blobs, Chapter 4 shows that error in code, and Chapter 19 gives every managed identity exactly the data-plane roles it needs and nothing more.

```
           Control plane (manage)              Data plane (use)
portal / az / Bicep / SDK                 your app code / SDK clients
          |                                          |
          v                                          v
https://management.azure.com           https://<account>.blob.core.windows.net
(Azure Resource Manager)                https://<vault>.vault.azure.net
          |                                 <namespace>.servicebus.windows.net
          v                                          |
create, configure, scale, delete         read, write, send, receive, query
```

> **NOTE**
>
> Because every management tool goes through the same ARM API, anything you can click in the portal you can also script with the Azure CLI or describe in a Bicep template. We often show the portal first to build intuition, then the CLI for repeatability, and finally Bicep for the version we actually keep.

Azure's Global Infrastructure in Brief

Azure runs in datacenters around the world, grouped into **regions**. A region is a set of datacenters in a geographic area connected by a low-latency network, such as East US 2 (`eastus2`), Central US (`centralus`), West Europe, or Southeast Asia. When you create most resources, you choose a region, and your data stays in that region unless you configure replication.

Many regions also offer **availability zones**: physically separate locations within the region, each with independent power, cooling, and networking. Services can be *zone-redundant* (the platform spreads your resource across zones automatically), *zonal* (you pin a resource to one zone), or simply *regional* (the service manages placement without zone guarantees). Not every region supports zones, and not every service supports zone redundancy in every region, so check the current documentation when it matters.

Regions are grouped into **geographies** that respect data residency boundaries, and many regions have a paired region (some newer regions have none). Separate **sovereign clouds** for governments are outside the scope of this book.

Chapter 2 explains how to choose a region for the Halcyon Portal and why we use East US 2. Chapter 27 returns to zones and multi-region designs when we talk about reliability.

A Quick Tour of the Azure Portal

The Azure portal at `https://portal.azure.com` is the web interface for everything in Azure. You will use it to explore, to check what a command did, and to read metrics and logs. You will *not* use it as the source of truth for infrastructure; that job belongs to Bicep, starting in Chapter 5. Still, knowing your way around saves time.

The parts you will use most

- **The global search bar** at the top. Type a resource name, a service name ("Key Vault"), or a resource group, and jump straight there. This is faster than any menu.
- **Resource blades.** Each resource opens in a page (historically called a blade) with a left-hand menu: **Overview**, **Activity log**, **Access control (IAM)**, **Tags**, a **Settings** group specific to the service, and a **Monitoring** group with metrics, alerts, and logs.
- **Access control (IAM)** on every resource, resource group, and subscription. This is where you view and add Azure role assignments, as Chapter 3 explains.
- **Activity log.** A record of control-plane operations: who created, changed, or deleted what, and when. It is the first place to look when "something changed."
- **Cloud Shell**, the terminal icon in the top bar. It opens a browser-based Bash or PowerShell session that is already signed in to your account and has the Azure CLI, Bicep, Git, and the .NET SDK preinstalled. It needs a small storage setup the first time, and the portal walks you through it.

> **TIP**
>
> If you cannot install tools on your machine yet, Cloud Shell is enough to follow Chapters 2 through 5. Every CLI command in this book uses Bash line continuations (a trailing backslash), which work unchanged in Cloud Shell's Bash mode.

The portal changes frequently. Menu items move, blades get renamed, and new previews appear. When we describe a portal path such as **Settings > Environment variables**, treat it as a guide, not a guarantee: if the label differs, use the search box inside the resource menu, which filters menu items by name.

Meet Halcyon Logistics

Throughout this book we build a single application for a fictional company. **Halcyon Logistics** is a freight and warehousing company with a few hundred employees spread across operations, finance, human resources, and IT. Its email domain is `halcyon.example.com`, and its public website lives at `www.halcyon.example.com`.

Today, Halcyon handles leave requests with spreadsheets and email, and supporting documents are emailed as attachments to a shared mailbox. Managers lose track of approvals, and nobody is sure who can see which documents.

The company has decided to build an internal employee portal on Azure. The work falls to a small development team and the platform engineer, **Grace Liu** (employee number HL-1008), who owns the Azure setup. When an example needs a person at the keyboard running commands, it is usually Grace.

The Halcyon Portal

The **Halcyon Portal** is the employee-facing web application at the center of the book. Employees sign in with their Microsoft Entra ID work accounts (Entra ID is the identity service formerly called Azure Active Directory; Chapter 3 explains the rename). The portal lets them:

- View their profile and their leave balances for the year.
- Submit leave requests, which are routed to their manager for approval.
- Approve or reject the requests of their direct reports, if they are managers.
- Submit HR requests: employment letters, address changes, payroll questions, benefits questions, and other topics.
- Upload supporting documents (PDF, PNG, or JPEG up to 10 MB), which are checked before HR can open them.
- Receive in-app notifications, for example when a request needs their approval or has been decided.
- Use HR admin views to work through open HR requests, if they are in the HR team.

We use a small, fixed set of sample employees everywhere in the book so that examples, test data, and outputs line up from chapter to chapter.

Number	Name	Job title	Department	Portal roles
HL-1001	Amara Osei	Head of Operations	Operations	Employee, Manager
HL-1002	Jonas Weber	Operations Supervisor	Operations	Employee, Manager
HL-1003	Lucia Romero	Dispatcher	Operations	Employee
HL-1004	Kenji Watanabe	Warehouse Lead	Operations	Employee
HL-1005	Fatima Haddad	HR Manager	Human Resources	Employee, Manager, HRAdmin
HL-1006	Ravi Menon	HR Specialist	Human Resources	Employee, HRAdmin
HL-1007	Tom Becker	Finance Manager	Finance	Employee
HL-1008	Grace Liu	Platform Engineer	IT	Employee

Jonas reports to Amara; Lucia and Kenji report to Jonas; Ravi reports to Fatima. Employees without a manager are approved by the HR Manager, and the HR Manager's own requests go to the Head of Operations. Nobody approves their own request.

One leave request deserves an early introduction because it appears in many chapters: **LR-2026-0008**. Kenji Watanabe asks for annual leave from Monday, November 23 to Friday, November 27, 2026. That is five weekdays, but Thursday, November 26 is Thanksgiving Day, a company holiday, so the request costs him **4 working days**. It is pending approval by Jonas. We will follow this request as

it is validated in code, stored in Azure SQL Database, published as an event on Azure Service Bus, turned into a notification in Azure Cosmos DB, written to an audit log, and traced in Application Insights.

Whenever an example depends on the current date, we use **Thursday, October 1, 2026** (UTC) as "today."

The Target Architecture

Here is where we are going. Do not worry if many of these names are new; each one gets a chapter. The diagram shows the `dev` environment; `prod` mirrors it with `-prod` names and stronger tiers.



```
Gateway     apim-halcyon-portal-dev (partner API)            [Ch 23]
Network     vnet-halcyon-portal-dev + private endpoints      [Ch 24]
Governance  Defender for Cloud, Policy, locks, budgets       [Ch 25-26]
Infra code  infra/main.bicep + modules                       [Ch 5+]
```

Read it as three flows.

1. **The request flow.** An employee signs in and uses the web app. The app reads and writes relational data (employees, leave requests, HR requests, document records) in Azure SQL Database, reads the employee's notification feed from Cosmos DB, and reads settings and feature flags from App Configuration, with the one external API key kept in Key Vault.
2. **The event flow.** When something important happens, such as a submitted or decided leave request, the web app publishes an event to the Service Bus topic `portal-events`. Three subscriptions fan the event out to three functions: one creates notifications, one writes an audit row, and one appends approved leave to a monthly payroll export file.
3. **The document flow.** The browser uploads a file directly to Blob Storage using a short-lived, narrowly scoped upload URL. Event Grid notices the new blob and triggers a function, which marks the document as scanning and queues it. A worker running in Container Apps checks the file, moves accepted files to a protected container, updates the database, and publishes a `DocumentAccepted` event.

Notice what is *not* in the diagram: passwords and connection strings with keys. Every arrow is authenticated with Microsoft Entra ID, using the developer's own sign-in on a laptop and a managed identity in Azure. The only secrets in the whole system are an API key for a fictional external HR system, kept in Key Vault, and a payroll SFTP password used in Chapter 18 as a rotation example. Getting to that state is one of the book's main goals.

Why these services?

Other teams could choose differently; here is our short reasoning:

- **App Service** for the web app, because it is the most direct managed host for an ASP.NET Core application, with deployment slots, built-in scaling, and simple custom domains.
- **Azure Functions** for small event handlers and a scheduled job, because they react to messages and timers without us writing hosting or polling code.
- **Container Apps** for the document worker, because it is a long-running background process that should scale out when the queue fills and scale to zero when it is empty.
- **Azure SQL Database** for core records, because leave requests and approvals are relational, need transactions, and benefit from constraints.
- **Cosmos DB** for notifications, because the access pattern is "one employee's recent items," the volume is spiky, and old items should expire automatically.
- **Service Bus** for business events, because we need reliable delivery, retries, dead-lettering, and filtered fan-out to several consumers.

- **Event Grid** for storage events, because it pushes "a blob was created" notifications without polling.

Naming Things

Names in Azure are surprisingly important. Some can never be changed after creation, some must be unique across all of Azure, and some appear in URLs that your users see. We follow a naming pattern based on the abbreviations recommended in Microsoft's Cloud Adoption Framework:

```
<abbreviation>-<workload>-<environment>

app-halcyon-portal-dev       web app, dev
kv-halcyon-portal-prod       Key Vault, prod
sthalcyonportaldev           storage account (no hyphens allowed)
```

The workload is `halcyon-portal` (or a component name such as `docworker`), the environment is `dev` or `prod`, and everything is lowercase. Resources whose names forbid hyphens, such as storage accounts and container registries, simply drop them. Here is the short version of the naming table; Chapter 2 gives the complete list.

Resource	Abbreviation	Dev name
Resource group	`rg`	`rg-halcyon-portal-dev`
App Service plan	`asp`	`asp-halcyon-portal-dev`
Web app	`app`	`app-halcyon-portal-dev`
Function app	`func`	`func-halcyon-portal-dev`
Storage account (data)	`st`	`sthalcyonportaldev`
SQL logical server	`sql`	`sql-halcyon-portal-dev`
Cosmos DB account	`cosmos`	`cosmos-halcyon-portal-dev`
Service Bus namespace	`sbns`	`sbns-halcyon-portal-dev`
Key Vault	`kv`	`kv-halcyon-portal-dev`
Log Analytics workspace	`log`	`log-halcyon-portal-dev`

> **NOTE**
>
> Several of these names must be globally unique across all Azure customers, because they become part of a public DNS name, such as `sthalcyonportaldev.blob.core.windows.net`. If you follow along and a name is already taken, append a short suffix such as `-k7` (or `k7` where hyphens are not allowed) and use it consistently everywhere that name appears.

Environments and Tiers

We build two environments: `dev`, which you will create as you read, and `prod`, which we describe and define in Bicep but which you do not need to deploy to learn the material. A `test` environment is discussed where it matters but not built.

The two environments use the same services with different tiers. Dev favors low idle cost: a Basic App Service plan, a serverless SQL database that auto-pauses, serverless Cosmos DB, the Consumption plan for functions, and a worker that scales to zero. Prod favors reliability and isolation: Premium v3 with a staging slot, provisioned SQL compute, autoscale Cosmos DB throughput, Elastic Premium functions, Premium messaging, private networking, and locks. Tier names are not prices; model your own costs before deploying anything you intend to keep.

How to Follow Along

This is a hands-on book. You get the most out of it by creating the resources yourself, running the code, and breaking things on purpose. Here is what you need.

An Azure subscription

You need access to an Azure subscription where you can create resource groups and assign roles. There are three common situations:

- **A new personal account.** Microsoft offers a free account with some free services and introductory credit for new customers. The details change over time, so check the current free account and free-offer terms before you sign up, and read what happens when the trial period ends.
- **A Visual Studio subscription or similar benefit.** Some developer programs include monthly Azure credit. Check your benefit's current terms.
- **Your employer's subscription.** Ask for a sandbox subscription or at least a dedicated resource group where you have the Owner role, or Contributor plus the ability to create role assignments. Chapter 3 explains why role assignments matter so much.

> **WARNING**
>
> Resources cost money while they exist, even if nobody uses them. Some services in this book bill by the hour regardless of traffic, and API Management in particular can take a long time to create. Set a budget alert as soon as you have a subscription (Chapter 26 shows how, and the portal's **Cost Management > Budgets** page is quick to use), and delete what you are not using.

Tools on your machine

- **The .NET 10 SDK.** All C# code in this book targets .NET 10, the current long-term support release. Apps written for .NET 8 (also LTS) work the same way with the services we use, but .NET 8 reaches end of support in November 2026, so new applications should start on .NET 10.

- **The Azure CLI** (`az`). Chapter 4 covers installation and the essential commands. Cloud Shell works too.
- **An editor.** Visual Studio, Visual Studio Code with the C# Dev Kit, or JetBrains Rider all work. Visual Studio Code's Bicep extension is excellent for Chapter 5 onward.
- **Git and a GitHub account**, for the CI/CD chapter. Our sample repository is `halcyon-logistics/halcyon-portal` with a default branch named `main`.
- **Azure Functions Core Tools** and **Docker** (or another container tool), for Chapters 8 and 9. You can install these when you get there.

Cleaning up

Everything we create for the dev environment lives in one resource group, `rg-halcyon-portal-dev`. That gives you a one-line cleanup command that deletes the group and every resource inside it:

```
az group delete --name rg-halcyon-portal-dev --yes --no-wait
```

The `--yes` flag skips the confirmation prompt, and `--no-wait` returns immediately while Azure deletes resources in the background. Deleting a resource group is permanent for most resources. A few services, such as Key Vault, keep deleted items in a soft-deleted state for a retention period, which means a vault name stays reserved until the retention period ends or the vault is purged. Chapter 18 explains the consequences. Because the whole environment is described in Bicep from Chapter 5 onward, you can recreate it in minutes, so do not hesitate to delete the group at the end of a study session.

> **TIP**
> Get into the habit of asking, at the end of each chapter, "what is running right now, and do I still need it?" Run `az resource list --resource-group rg-halcyon-portal-dev --output table` to see the answer.

A note for Python and JavaScript developers

The main path in this book is C# on .NET 10, because the Halcyon team is a .NET shop. The Azure concepts, however, are language-neutral: the same resources, roles, identities, and service behaviors apply whatever language you write. The Azure SDKs for Python (packages such as `azure-identity`, `azure-storage-blob`, `azure-servicebus`, and `azure-cosmos`) and for JavaScript and TypeScript (`@azure/identity`, `@azure/storage-blob`, `@azure/service-bus`, `@azure/cosmos`) follow the same design guidelines as the .NET SDK. A credential object, a client per service, and similar method names appear in all of them. Where it helps, we show short Python or Node.js equivalents next to the C# code.

Here is a taste of that consistency: creating a blob client with a passwordless credential looks almost the same in all three languages. The C# version comes first.