GDCD ON AZURE

BOOK 1



GOVERNANCE-DRIVEN CONTRACT DESIGN

Governing Azure by Contract

A layer-by-layer guide to a governed Azure platform.

Most Azure estates are not designed. They accumulate: a subscription here, an Owner role there, a public endpoint nobody remembers. This guide starts from the business outcome, turns it into guarantees, and writes those guarantees as contracts at every layer of Azure: the tenant, the management group tree, subscriptions per environment, roles, policy, network, security, visibility and cost. The result is a governed landing zone that any workload, including AI, can join safely.

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

CONTENTS

00 Why This Book

PART I · OUTCOME FIRST

01 Govern Before You Build

02 The Tenant Is the First Door

PART II · THE STRUCTURE

03 The Management Group Tree

04 Subscriptions: Dev, Test, Prod and the Vending Machine

PART III · ACCESS AND RULES

05 Roles: Who May Do What, and for How Long

06 Policy Is the Contract That Runs

PART IV · GUARANTEE LAYERS

07 Network: Private by Default

08 Security Posture You Can Watch

09 Visibility: One Place to See Everything

10 Cost: Every Number Adds Up

PART V · OPERATE

11 Deploy by Contract: Infrastructure as Code

12 The Governance Operating Model

A Appendix: Templates

BEFORE YOU BEGIN

Why This Book



Most Azure estates are not designed. They accumulate. A team needs a database, a subscription is opened outside any agreed process, and a year later the organization owns hundreds of resources that nobody can fully explain. Nobody chose that outcome. It is simply what happens when building starts before governing.

This book is about doing it the other way around. It shows how to govern a full Azure estate layer by layer: the tenant, the management group tree, subscriptions per environment, roles, policy, network, security, visibility and cost. Each layer is treated as a door, and each door gets a contract: a short, written, checkable agreement about who may cross, what may cross, what must be true afterward and who finds out.

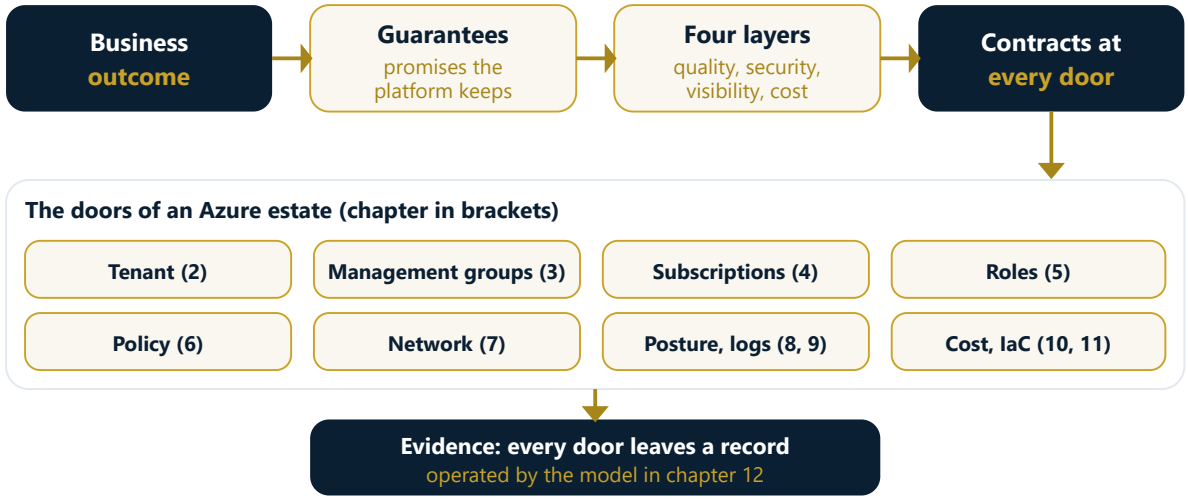
The method behind it is Governance-Driven Contract Design (GDCD). The foundational GDCD book, *The Contract Decides: Governing AI at Every Door*, introduced it for AI agents. This book applies the same spine to the cloud platform underneath. In production estates, the two are not separate problems. An AI workload is only as governed as the subscription, identity and network it runs in.

The spine of the method

GDCD always moves in the same order. Start from the business outcome. Turn the outcome into guarantees, the promises the platform must keep. Spread the guarantees across four layers: quality, security, visibility and cost. Then write each guarantee as a contract at the door where it can be enforced, and make the platform enforce it, not goodwill.

FIGURE 0.1

The GDCD spine, applied to Azure



Outcome first, doors last. The doors are where Azure gives us enforcement points. The book walks them in the order a platform team should build them.

The order matters. Teams that start from tools ("we need a firewall", "we should turn on Defender") end up with good products wired to no purpose. Teams that start from the outcome ("customer data never leaves the approved regions", "nobody holds standing admin rights in production") can trace every setting back to a reason. When an auditor, a new CISO or a new platform lead asks "why is this here?", the answer is already written down.

> *The model proposes. The contract decides. On a cloud platform, the "model" is every person, pipeline and agent that wants to change the estate.*

Who this book is for

It is written for the people who answer for an Azure estate: the CIO who signs the budget, the cloud architect who draws the platform, the platform team lead who runs it, the CISO who must prove it is safe, and the FinOps lead who must explain every line of the bill. Each chapter opens with its point in plain words so that a leader can read the first page and know what to ask for. The rest of the chapter is the engineering detail a platform team needs to build it.

You do not need to read it front to back. The structure follows how a platform is built:

Part	Chapters	What you will be able to do

I. Outcome first	1 to 2	Turn business outcomes into platform guarantees, and secure the tenant, the first door.
II. The structure	3 to 4	Draw the management group tree and the subscription map, and vend governed subscriptions on request.
III. Access and rules	5 to 6	Design roles that expire, and write policy that runs as a contract.
IV. Guarantee layers	7 to 10	Make the network private by default, watch security posture, see everything in one place and make every number add up.
V. Operate	11 to 12	Deploy only through code and pipelines, and run governance as a team with a 30-60-90 day rollout.
Appendix	After 12	Templates: the management group and subscription map, role matrix, policy starter set, tag dictionary and readiness checklist.

Principle over product

Microsoft renames things. Azure Active Directory became Microsoft Entra ID. Azure Security Center became Microsoft Defender for Cloud. The landing zone tooling has been rebuilt more than once. If this book taught button clicks, it would age in months. Instead, every chapter teaches the pattern first, then shows how Azure implements it today. The pattern is the durable part. The product name is the current address of the pattern.

Where the book names a product, a built-in role or a policy effect, it uses the name in Microsoft's documentation at the time of writing. Product names, built-in roles and features were checked against Microsoft documentation in November 2026. The guidance follows the Microsoft Cloud Adoption Framework and the Azure landing zone reference architecture, because they encode many years of field experience and because a platform that follows them is easier to hand over, audit and support. Where we depart from the reference, we say so and explain why.

The book assumes the Azure global cloud and one corporate tenant. Sovereign clouds such as Azure Government, servers and clusters outside Azure managed through Azure Arc, and estates operated by a partner through Azure Lighthouse follow the same patterns, but service availability and some features differ, so check the documentation for your cloud before you rely on a detail.

WHAT "GOVERNED" MEANS IN THIS BOOK

An estate is governed when, for every subscription, you can answer five questions in under a minute: who owns it, what it may contain, who may change it and for how long, where its logs go, and what it may cost. Every chapter adds the controls that make one of those answers true and provable.

How to use the chapters

Each chapter ends with a short contract checklist: three to six items you can tick against your own estate. Taken together, those checklists form a readiness review you can run against an existing estate. Each chapter also points to a live, interactive version of its diagrams at hub.jcardena.com/azure, where you can click through the management group tree, the role model and the policy set.

The companion book, *AI Foundry by Contract*, builds Microsoft Foundry on top of the platform described here. If your organization is being asked to deliver AI on Azure, read this book first. Every AI guardrail in the second book assumes the tenant, tree, subscriptions, roles and policy from this one.

One promise runs through every page: trust, earned at every layer. A platform earns trust the same way a person does, by keeping small promises consistently and leaving a record that it did.

CONTRACT CHECKLIST

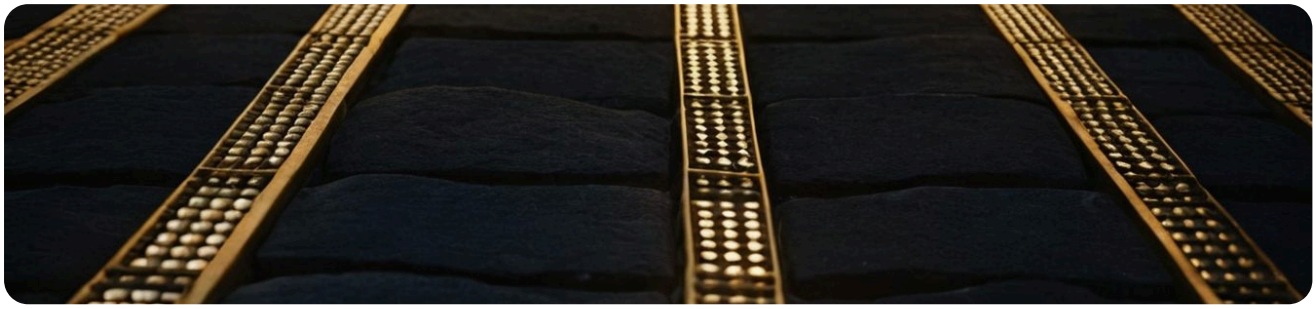
- We can name the business outcomes our Azure platform exists to protect.
- Each outcome has at least one written guarantee with a named owner.
- We know which door (tenant, tree, subscription, role, policy, network, pipeline) enforces each guarantee.
- For any subscription, we can answer owner, contents, access, logs and cost in under a minute.

See it live: hub.jcardena.com/azure

PART I · OUTCOME FIRST

CHAPTER ONE

Govern Before You Build



Governance added after the fact is always more expensive than governance designed in. Not a little more expensive. Every workload built without guardrails becomes a migration later: resources to move, permissions to unwind, public endpoints to close while users depend on them. The cheapest moment to decide who may deploy, where data may live and what things may cost is before the first resource exists.

Many inherited estates started the other way. A project needed results quickly. The team got an Owner role on a fresh subscription, built something that worked, and moved on. The next team copied the pattern. Two years later there are dozens of subscriptions with no common structure, logs that stay where they were born, and a monthly bill that nobody can split by product. The outcome followed from missing platform controls: the platform had no contract, so every team wrote its own.

Why Azure estates drift

Azure makes it easy to start. That is a strength. It also means that every default you do not change becomes your policy. New subscriptions land under the tenant root management group unless you set a different default. Any user can create a management group unless you require authorization for it. A person who creates a subscription often ends up as its owner. None of these defaults is wrong for a single team. All of them are wrong for an enterprise, and they compound quietly.

Drift has three faces that leaders recognize:

- **Access drift.** Owner and Contributor assignments pile up on individual people, at broad scopes, with no end date. Nobody remembers why half of them exist.
- **Configuration drift.** One workload uses private endpoints, the next exposes a storage account to the internet, and both passed their own review.

- **Cost drift.** Spend grows faster than any owner can explain, because resources carry no owner, cost center or environment tag.

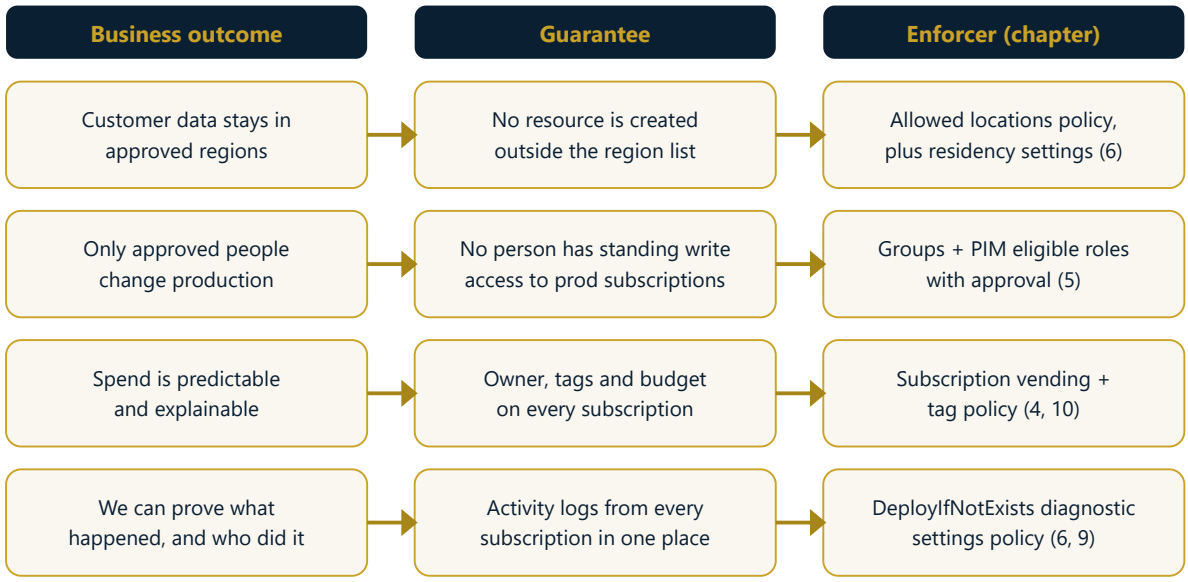
Each of these is a broken promise the business never knew it was relying on. GDCD names that promise first, then makes the platform keep it.

From outcome to guarantee to enforcer

The method in this book starts with a short conversation with the business, not with a product list. What must never happen? What must always be true? What must we be able to prove? The answers are business outcomes. We then rewrite each one as a guarantee: a precise statement the platform can keep. Finally, we name the enforcer, the Azure mechanism that keeps the guarantee without relying on anyone's memory.

FIGURE 1.1

Four outcomes, traced to the Azure door that enforces them



A row with no enforcer is a wish. Mark it, or remove it.

Read left to right in the meeting, right to left in the audit. Leaders own the first column. The platform team owns the third. The guarantee in the middle is the contract between them.

Notice that none of the guarantees mentions a product. "No resource is created outside the region list" will still be the right guarantee if the enforcer is renamed or replaced. That is the principle-over-tool rule from the preface, applied to a real decision.

The four guarantee layers

Once the outcomes are written, sort the guarantees into the four layers that run through the whole GDCD series. Each layer has a smallest unit and a silent failure it refuses.

Layer	What it guarantees on Azure	The silent failure it refuses	Chapters
Quality	Changes arrive through tested code and pipelines, and promotion from dev to test to prod is repeatable.	A manual fix in production that nobody can reproduce.	4, 11
Security	Identities are strong, access is minimal and temporary, networks are private by default.	A forgotten Owner or a public endpoint that nobody chose.	2, 5, 7, 8
Visibility	Every subscription sends its logs to one place, and every alert reaches a person.	An event that happened but left no record anyone could find.	9
Cost	Every resource has an owner and a cost center, every subscription has a budget.	The bill line nobody can explain.	10

Contracts, the policy and role rules in Part III, are not a fifth layer. They are how all four layers are enforced. That is why this book spends two full chapters on roles and policy before it reaches the network.

What "before you build" really means

Governing first does not mean a six-month design phase before anyone may deploy. That approach fails in a different way: teams go around the platform because it is not there yet. The pattern that works in practice is a thin, enforced foundation first, then fast delivery on top of it.

1. **Secure the tenant** (Chapter Two): emergency access accounts, strong authentication, separate admin identities.
2. **Draw the tree** (Chapter Three): management groups that match workload types, not the org chart.
3. **Open the vending machine** (Chapter Four): a request form that produces a governed subscription with owner, tags, budget and network already in place.
4. **Bind access and rules** (Chapters Five and Six): roles assigned to groups and made temporary, and a starter policy set running in audit first, then enforced.

That foundation is deliberately small. The guarantee layers in Part IV and the operating model in Part V harden it over time. What matters is that the first workload lands inside a governed subscription, so the second and the fiftieth do too.

A COMMON INHERITED ESTATE

A platform team inherits an estate with many subscriptions, most sitting directly under the tenant root. The team does not start by moving anything. It first creates the management group tree, sets the default management group for new subscriptions to Sandbox, and assigns a small policy set in audit mode at the top of the tree and directly to the existing subscriptions, until they move into the tree. Within days, the compliance view shows exactly where the debt is. The estate did not change, but for the first time it could be measured. Chapter Twelve turns that first step into a 30-60-90 day rollout.

Do not govern by document alone. A wiki page that says "only deploy to approved regions" is a wish. A policy assignment with the Deny effect on the landing zones management group is a contract. Write the document for people, and make sure every rule in it names its enforcer.

The cost of governing late

We do not quote a universal number for it, because it depends on the estate. The pattern, though, is consistent. Late governance arrives as three separate bills: one to build without guardrails, one to find out what was built, and one to fix it while it runs in production with users on it. Early governance costs once, and most of that cost is automation the platform team reuses for every later workload.

> *Guardrails are cheapest before the first resource exists. After that, every one of them is a migration.*

There is also a cost that never appears on a bill: trust. When a CISO cannot say who has admin rights in production, or a CFO cannot say which product drove last month's spend, confidence in the whole cloud program drops. A governed platform answers those questions in minutes. That is the real business outcome of everything that follows. Trust, earned at every layer.

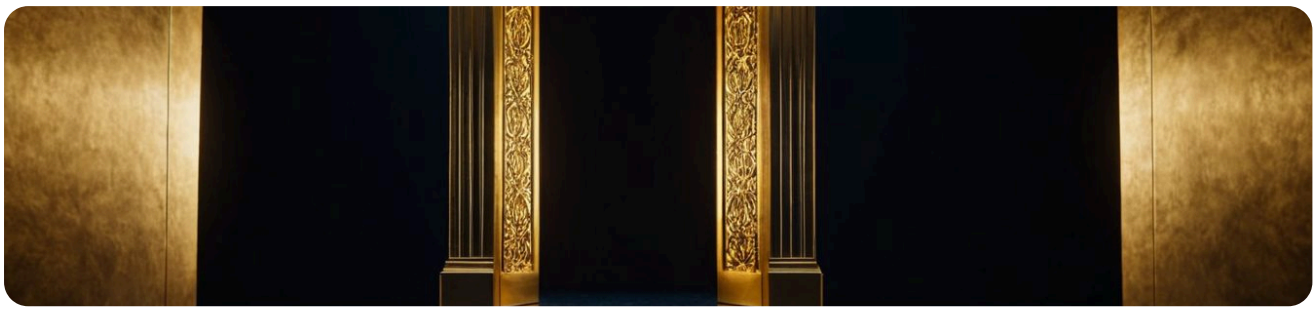
CONTRACT CHECKLIST

- We have written three to six business outcomes for the platform and agreed them with their owners.
- Each outcome is rewritten as a product-neutral guarantee.
- Each guarantee names an Azure enforcer and the chapter that builds it, or is marked as a wish.
- We have set a default management group for new subscriptions so none lands under the tenant root.
- We have a thin foundation plan (tenant, tree, vending, roles, policy) before the next workload is onboarded.

See it live: the governance map at hub.jcardena.com/azure/governance.html

CHAPTER TWO

The Tenant Is the First Door



Every Azure action starts with a sign-in. Before anyone touches a subscription, a policy or a network, Microsoft Entra ID has already decided who they are and how sure it is. If that decision is weak, nothing built above it can be strong. The tenant is the first door, and it is the one that every other door trusts.

A Microsoft Entra tenant is the identity boundary of the organization. Every subscription trusts exactly one tenant, and the management group tree from Chapter Three lives inside it. All subscriptions in a management group must trust the same tenant. So the first platform decision is not about Azure resources at all. It is about how many tenants you have, and how strongly each identity in them proves itself.

One tenant, unless a requirement forces more

Microsoft's landing zone guidance is clear: use a single Microsoft Entra tenant, usually the corporate tenant where people already sign in for email and collaboration, and create more tenants only when a requirement cannot be met in the corporate one. The reasons are practical. Azure landing zones are deployed per tenant, so every extra tenant means another tree, another policy set and another set of logs. Licenses for Microsoft Entra ID P1 or P2 do not span tenants. Many Azure services integrate natively with identities in their home tenant and offer limited support for guests from other tenants.

Separation inside one tenant is achieved with structure, not with more tenants: management groups and subscriptions for resources, and administrative units for delegating user and group administration. Sandbox subscriptions give teams room to experiment without a separate "test" tenant for Azure. A dedicated tenant for testing identity configuration itself can still make sense, but it should not hold Azure workloads.

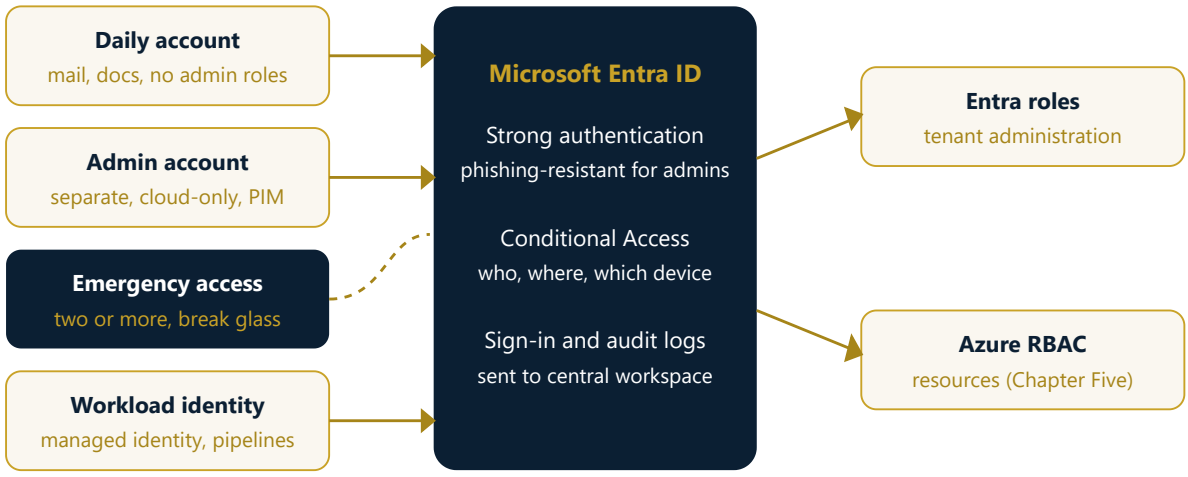
Moving subscriptions between tenants is not a quick fix. Transferring a subscription to another directory removes existing role assignments and requires careful pre and post work. Microsoft notes it is often easier to rebuild the workload in a new subscription in the target tenant. Decide the tenant before you build, not after.

Four kinds of identity, four contracts

Inside the tenant, we separate identities by what they are for, because each kind needs a different contract. Mixing them is one of the most common identity weaknesses in inherited estates.

FIGURE 2.1

The tenant door: every identity passes the same gate, under its own contract



Dashed: emergency accounts skip blocking Conditional Access and alert on every use.

One gate, four contracts. The gate is the same for everyone. What differs is how strongly each identity proves itself, which roles it may hold, and how its use is watched.

Identity	Contract
Daily account	Holds no administrative roles in Entra or Azure. Protected by multifactor authentication and Conditional Access like every user.
Admin account	A separate, cloud-only account used only for administration. Holds roles as eligible through Privileged Identity Management, activated when needed. Uses phishing-resistant authentication.
Emergency access	Two or more cloud-only accounts on the onmicrosoft.com domain, with passkey (FIDO2) or certificate-based authentication. Never used day to day, alerted on every sign-in.

Workload identity	Managed identities for Azure resources and federated credentials for pipelines. No shared passwords, no user accounts acting as service accounts.

Strong authentication is no longer optional

Microsoft now enforces multifactor authentication for Azure sign-ins. In the first phase, accounts signing in to the Azure portal, the Microsoft Entra admin center and the Intune admin center must complete MFA. In the second phase, accounts that create, update or delete resources through Azure CLI, Azure PowerShell, infrastructure as code tools, SDKs and the REST API must complete MFA too. Workload identities such as managed identities and service principals are not affected. User accounts used as service accounts are, which is one more reason to retire them.

Mandatory MFA is a floor, not a design. The contract for an enterprise tenant goes further: Conditional Access policies that require MFA for all users, phishing-resistant methods for administrators, and blocking of legacy authentication. Conditional Access needs Microsoft Entra ID P1 or P2. Tenants without it should at least turn on security defaults.

Workload identities cannot perform MFA, so their tenant contract is different. Conditional Access for workload identities can block a single-tenant service principal when it requests a token from outside known IP ranges, or when Microsoft Entra ID Protection rates it as risky. It needs Workload Identities Premium licenses, and it does not cover managed identities, which have no credential that anyone stores or handles. For people, ID Protection with Microsoft Entra ID P2 feeds user and sign-in risk into Conditional Access, so a risky sign-in can be blocked or forced through a secure password change.

Emergency access: the door for the worst day

Every strong control can lock you out. A federation outage, a misconfigured Conditional Access policy, or approvers who have all left can leave nobody able to administer the tenant. Emergency access accounts exist for that day. Microsoft's guidance is specific, and we follow it as written:

- At least two accounts, cloud-only, on the onmicrosoft.com domain, not federated or synchronized.
- Passkey (FIDO2) or certificate-based authentication, different from the methods normal admins use.
- Global Administrator assigned as permanent active in PIM, not eligible.
- Excluded from Conditional Access policies that block or restrict sign-in, through a dedicated group.
- Credentials stored in separate secure locations, used from a secure workstation.

- Every sign-in alerted, and the accounts validated at least every 90 days.

The alert is what turns an emergency account from a hidden risk into a governed door. Send Entra sign-in logs to the central Log Analytics workspace (Chapter Nine) and create a log alert rule that fires on any result:

```
// Any sign-in by an emergency access account raises a critical alert
SigninLogs
| where UserId == "<object-id-1>" or UserId == "<object-id-2>"
| project TimeGenerated, UserPrincipalName, UserId, IPAddress, ResultType,
ResultDescription
```

> **THE DRILL THAT FINDS THE GAP**
>
> A quarterly emergency access drill works best as a calendar event with two people present. One signs in, one watches for the alert. The drill regularly catches drift that nothing else would: a Conditional Access policy added without the exclusion group, or an alert action group pointing at a mailbox that no longer exists.

Keep the most powerful role small

Global Administrator controls the tenant, and a Global Administrator can elevate access to manage every Azure subscription and management group. Microsoft recommends keeping Global Administrator assignments to a minimum, fewer than five is preferred, making admin roles eligible through PIM, requiring approval to activate the most critical ones, and notifying stakeholders on activation. Use the narrower Entra roles for daily work. The setting "Access management for Azure resources" should stay off for Global Administrators who do not need it, because it grants User Access Administrator at the root of the tree.

Admin devices and application consent

Strong sign-in is wasted on a compromised device. Administrators of the tenant and the platform work from privileged access workstations: hardened, managed devices used only for administration, which Conditional Access can require for admin roles through device compliance. Applications are the other quiet path into the tenant. Restrict user consent, for example to verified publishers and low-risk permissions, or turn it off, and turn on the admin consent workflow so users can request approval instead of working around the block. Decide who may create app registrations, because by default any member user can, and give every registration a named owner.

> *Every door above the tenant trusts the tenant. Make the first door the strongest.*

CONTRACT CHECKLIST

- All Azure subscriptions trust one corporate tenant, and every extra tenant has a written requirement.
- Administrators use separate, cloud-only admin accounts with phishing-resistant authentication.
- Two or more emergency access accounts exist, are excluded from blocking Conditional Access, and alert on every sign-in.
- The emergency access drill has run in the last 90 days.
- Global Administrator has fewer than five assignments, all eligible through PIM except emergency accounts.
- No user account is used as a service account for automation.

See it live: the roles map at hub.jcardena.com/azure/governance-roles.html

PART II · THE STRUCTURE

CHAPTER THREE

The Management Group Tree



A management group tree is an org chart for risk, not for people. Each branch holds subscriptions that need the same rules, and every rule assigned to a branch flows down to everything beneath it. Draw the tree around what workloads need, keep it shallow, and most of your policy and access design falls into place on its own.

Management groups are the governance scope above subscriptions. Policy assignments and role assignments made on a management group are inherited by every child management group, subscription, resource group and resource. That inheritance is the whole point. One assignment on the right branch replaces hundreds of per-subscription settings, and nobody below the branch can quietly opt out.

The facts that shape the design

A few platform facts from Microsoft's documentation set the boundaries for any tree:

- Each directory has one root management group, shown by default as **Tenant root group**, whose ID is the tenant ID. It cannot be moved or deleted.
- A tree supports up to six levels of depth, not counting the root or the subscription level.
- Every management group except the root, and every subscription, has exactly one parent.
- New subscriptions land under the root management group unless a default management group is set, as described below.
- By default, any principal in the tenant can create management groups, unless you require authorization in the hierarchy settings.

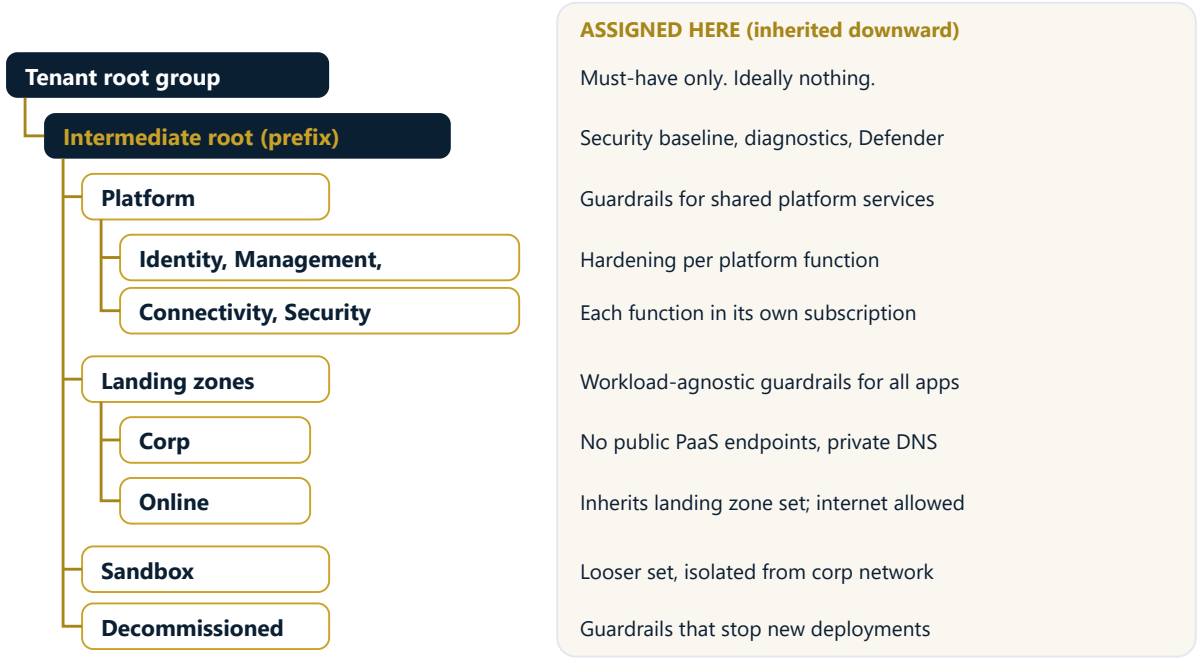
Six levels is a limit, not a target. The Cloud Adoption Framework recommends keeping the hierarchy reasonably flat, ideally no more than three to four levels. Every extra level is one more place where an inherited rule can surprise someone.

The landing zone tree

The Azure landing zone reference architecture gives a tree that fits most enterprises with little change. Our platform team starts every new tree from it, because it encodes field experience and because the next platform engineer will recognize it.

FIGURE 3.1

The Azure landing zone tree, and what is assigned at each level



Left: the tree. Right: the contract on each branch. Workload subscriptions live under Corp or Online. Platform subscriptions live under their platform group. Sandbox and Decommissioned sit directly under the intermediate root. A Local group for Azure Local hybrid workloads can be added under Landing zones when needed.

Each group has a clear job:

Management group	Holds	Purpose of its rules
Intermediate root	Everything below, named with an organization prefix	Lets you avoid assigning anything at the tenant root, and gives a single top for the estate's baseline.
Platform	Identity, Management, Connectivity and Security groups	Each holds a dedicated subscription: identity services, the central Log Analytics workspace, the hub network, and security tooling such as Microsoft Sentinel.
Landing zones	Corp, Online, and optionally Local	Corp is for workloads that connect to the corporate network through the hub. Online is for workloads that need direct internet access or no virtual network.

Sandbox	Experiment subscriptions	A less restrictive policy set for exploration, isolated from corp and online landing zones.
Decommissioned	Cancelled landing zones	Parking for subscriptions on their way out, with guardrails that stop new use.

The default policy assignments in the Azure landing zone accelerators follow this shape. At the time of writing, for example, the Corp group carries a Deny assignment that disables public network access for PaaS services, and Sandbox and Decommissioned each carry their own guardrail initiative. The exact list evolves with each release, so treat the accelerator's current assignments as a starting point to review, not as a fixed standard.

Rules for a tree that lasts

DESIGN RULES WE APPLY

- **Group by workload archetype, not by department.** Departments reorganize. The rules a corp-connected workload needs do not.
- **Do not create management groups for dev, test and prod.** Environments are separate subscriptions inside the same archetype group. Chapter Four explains why, and Chapter Six covers the policy side.
- **Do not create management groups per region** unless a location-based legal requirement, such as data residency, demands it.
- **Assign policy at management groups, access at subscriptions.** Workload teams get roles at subscription or resource group scope. Platform teams may hold roles at management group scope, controlled through PIM.
- **Keep the root nearly empty.** Anything assigned at the tenant root applies to every resource in the directory.
- **Use tags, not depth, for cross-cutting views.** Cost center or product can be queried across the tree without adding levels.

The arithmetic behind the second rule is worth showing to anyone who wants environment groups. Thirty workloads, each with its own group plus children for development, test and production, produce 120 management groups. That is the example Microsoft uses in its own guidance, and every one of those groups is a place where rules can diverge between environments.

Lock the tree itself

The tree is a contract, so changes to it need their own guardrails. Two hierarchy settings matter from day one. First, require authorization for management group operations, so only principals with the right Azure RBAC permission can create groups. Second, set a default management group for new subscriptions. Microsoft suggests the Sandbox group as a good candidate, so that any new subscription arrives with a policy set already applied instead of landing under the root with none.

Build the tree from code like everything else (Chapter Eleven). A management group is a small resource deployed at tenant scope:

```
targetScope = 'tenant'

param prefix string = 'org'

resource landingZones 'Microsoft.Management/managementGroups@2021-04-01' = {
  name: '${prefix}-landingzones'
  properties: {
    displayName: 'Landing zones'
    details: {
      parent: {
        id: tenantResourceId('Microsoft.Management/managementGroups', prefix)
      }
    }
  }
}
```

Moves matter as much as creation. Moving a subscription between branches changes which policies apply to it, and Azure Resource Manager can cache hierarchy details for up to 30 minutes, so the portal may lag behind a move. Moves are made by the platform team through the pipeline, never by hand, and every move is visible in the activity log at management group scope.

MOVING AN INHERITED ESTATE INTO THE TREE

When an existing subscription moves from the root into Corp, it suddenly inherits a Deny on public endpoints. Existing resources keep running and show as non-compliant, and the next change to them is denied. Microsoft also documents cases where policy blocks the move itself because of resource properties inside the subscription. We therefore move subscriptions only after the policy set has run in audit against them and the owners have a plan, or after every Deny on the target branch has been set to DoNotEnforce, so the move changes what is measured, not what is allowed. Where a move must happen first, a time-boxed exemption with an expiry date covers the gap; for the blocked-move case, an exemption can be created before the move with its assignment scope validation set to DoNotValidate, a setting in preview at the time of writing. Chapter Six shows how exemptions work.

Draw the tree once, around risk. Then let inheritance do the repetitive work, every minute, without anyone remembering to.

CONTRACT CHECKLIST

- Our tree follows the landing zone shape: intermediate root, Platform, Landing zones, Sandbox, Decommissioned.
- The tree is no deeper than four levels below the root, and has no environment or department branches.
- Nothing is assigned at the tenant root without a written must-have reason.
- Management group creation requires authorization, and new subscriptions default to Sandbox.
- The tree is deployed and changed only through code.

See it live: the landing zone map at hub.jcardena.com/azure/governance-landing-zone.html

CHAPTER FOUR

Subscriptions: Dev, Test, Prod and the Vending Machine



A subscription is the unit of control in Azure. It is where access is granted, where a budget is set, where quotas apply and where the bill is split. Give each workload one subscription per environment, place them in the right branch of the tree, and let a vending machine create them. Then every new workload starts governed, in minutes, without a ticket queue.

The Cloud Adoption Framework calls this subscription democratization: subscriptions, not resource groups, are the primary unit of workload management and scale. That choice has a direct governance payoff. A subscription is a strong boundary for access, billing, quota and deployment. Access granted inside one subscription does not reach another, a budget alert names one owner, and a mistake in development cannot touch a production resource it cannot see. Two things cross it on purpose: roles inherited from a management group, and network paths built between subscriptions.

One subscription per workload per environment

Microsoft's guidance is that, ideally, each application environment has its own subscription. Development, test and production for the same workload become three subscriptions, all placed in the same archetype management group, Corp or Online. Sandbox is the one exception: exploratory work lives in subscriptions under the Sandbox group, with a looser policy set and no path into the corporate network.

FIGURE 4.1

A subscription map: environments are subscriptions, not branches

Workload	Archetype group	Development	Test	Production
Claims portal	Corp	sub-claims-dev	sub-claims-test	sub-claims-prod
Public website	Online	sub-web-dev	sub-web-test	sub-web-prod
Data platform	Corp	sub-data-dev	sub-data-test	sub-data-prod
Team experiments	Sandbox	sub-sbx-<team>, time-limited, no corp connectivity		

Same group, same guardrails, separate blast radius. Each row is promoted through identical rules, so what passes in test behaves the same in production. The subscription boundary keeps a development mistake out of production.

Why not management groups for each environment? Because Microsoft's guidance, and our experience, is that the development lifecycle environments should not have different guardrails. If development allows a public storage account and production denies it, the team only discovers the problem at the last step, after building around the wrong design. Keep the guardrails the same and let the subscription boundary provide the isolation. Where production genuinely needs an extra obligation, such as backup, add it at subscription scope during vending.

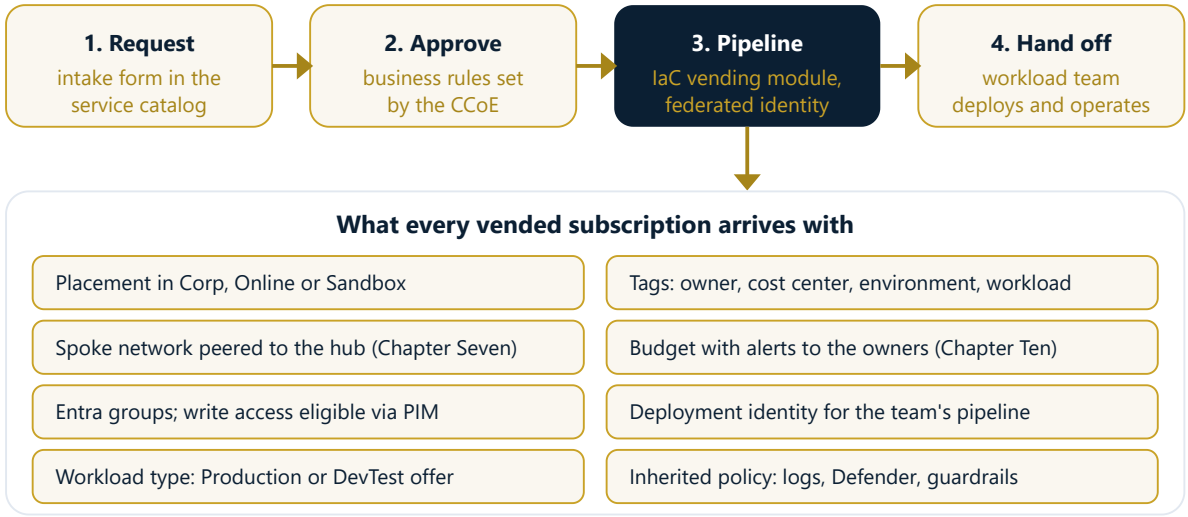
There are honest exceptions. Some designs share a single resource across environments, App Service deployment slots being a common example, because a slot lives on one App Service plan in one subscription. Microsoft's guidance allows a single subscription for several environments when they cannot be isolated, share the same operators and can use the same policies. Write the exception down and protect it with resource group scoped access through PIM.

The vending machine

Subscription vending is the platform mechanism that issues governed subscriptions on request. It gives workload teams one official front door, gives the platform team control of every subscription's starting state, and replaces weeks of tickets with a pipeline run. Three teams take part: the Cloud Center of Excellence sets the business rules and approval logic, the workload team requests, and the platform team's automation creates and configures the subscription before handing it over.

FIGURE 4.2

Subscription vending: from request to a governed subscription



The request is the contract. Everything the form asks for becomes a parameter, and every parameter becomes a guarantee the subscription carries from its first minute.

The intake form is where governance becomes cheap. Microsoft recommends gathering at intake the anticipated budget, the subscription owners, networking expectations, and the business criticality and confidentiality classification. Those answers decide placement in the tree, the network shape, the budget and who approves. Integrate the form with your existing IT service management tool, so approvals, audit history and reporting live where the organization already looks for them, and use an IP address management tool so that no two spokes overlap.

Three details are easy to miss. First, the request must say whether the workload is Production or DevTest, because the Dev/Test offer has lower rates for some resources but different terms, and is not available under every agreement type. Second, quotas apply per subscription and per region, for example vCPUs per virtual machine family, so a new subscription starts with default limits. The intake form should ask for the expected scale, and the vending pipeline should request the quota increases the workload needs before handover, so the first production deployment does not stall on a limit. Third, budgets are alerts, not hard limits. The vended budget tells owners when spend approaches the threshold; it does not stop resources. Chapter Ten covers the controls that do.

Vending as code

Microsoft publishes subscription vending modules for Bicep and Terraform as Azure Verified Modules. The pipeline, run from GitHub Actions or Azure Pipelines with a federated workload identity, calls the module with the parameters from the approved request. A shortened Bicep example:

```
targetScope = 'managementGroup'

// Parameter names as published for module version 0.8.0.
module claimsProd 'br/public:avm/ptn/lz/sub-vending:0.8.0' = {
  name: 'vend-claims-prod'
  params: {
    subscriptionAliasEnabled: true
    subscriptionAliasName: 'sub-claims-prod'
    subscriptionDisplayName: 'sub-claims-prod'
    subscriptionBillingScope: billingScope
    subscriptionWorkload: 'Production'
    subscriptionManagementGroupAssociationEnabled: true
    subscriptionManagementGroupId: 'org-corp'
    subscriptionTags: {
      owner: 'claims-product-owner'
      costCenter: 'CC-1042'
      environment: 'prod'
      workload: 'claims'
      dataClassification: 'confidential'
    }
    roleAssignmentEnabled: true
    roleAssignments: [
      {
        principalId: claimsReadersGroupId
        definition: 'Reader'
        relativeScope: ''
        principalType: 'Group'
      }
    ]
    budgetName: 'budget-claims-prod'
    budgetAmount: budgetFromRequest
    budgetThresholds: [ 80, 100 ]
  }
}

param billingScope string
param claimsReadersGroupId string
param budgetFromRequest int
```

Pin the module version your platform team has tested, and upgrade it deliberately. Note what the example does not do: no person receives standing write access. Contributor for the workload team is configured as an eligible assignment through PIM for Groups, as Chapter

Five describes, because elevated access to a new subscription should never be permanent by default. Creating subscriptions programmatically also requires the right billing role on your Enterprise Agreement, Microsoft Customer Agreement or partner agreement; check those prerequisites before the first run.

Names and tags are part of the contract

A name tells a person what a resource is. A tag tells a machine. Both need a standard that the platform enforces rather than requests. For names, we follow the Cloud Adoption Framework pattern of resource type, workload, environment, region and instance, for example *vnet-claims-prod-weu-001*. Some resource types restrict characters: storage account names allow only lowercase letters and numbers, so the same pattern becomes *stclaimsprodweu001*. For tags, keep the required set small and meaningful:

Tag	Meaning	Enforced by
owner	The accountable team or role, never a single person's name	Vending sets it on the subscription; policy inherits it to resources
costCenter	Where the spend is charged	Same: set once, inherited by policy
environment	dev, test, prod or sandbox	Set by vending, matches the subscription
workload	Short workload name	Set by vending, matches the subscription
dataClassification	public, internal, confidential, restricted	Required on resources holding data

Azure Policy has built-in definitions for both halves of this. *Inherit a tag from the subscription if missing* uses the Modify effect to copy a subscription tag onto resources, and remediation tasks can fix existing ones. *Require a tag on resources* uses Deny for tags that only the deploying team can know. Chapter Six explains how those effects behave, and the appendix holds a full tag dictionary.

> **Do not let tags carry security decisions.** Microsoft warns that people with elevated rights on a subscription can change tags. Tags are excellent for cost and ownership; security guardrails belong at management group scope, where workload teams cannot alter them.

> *A subscription that arrives governed stays governed. One that arrives empty becomes someone's private cloud.*

CONTRACT CHECKLIST

- Each workload has separate dev, test and prod subscriptions in the same archetype group, or a written exception.
- New subscriptions come only from the vending pipeline, with intake data stored in the service management tool.
- Every vended subscription has owner, cost center, environment and workload tags and a budget with alerts.
- No person receives standing write access in a vended subscription; elevated roles are eligible through PIM.
- Names follow one documented pattern, and tag rules are enforced by policy, not by review.

See it live: the landing zone map at hub.jcardena.com/azure/governance-landing-zone.html

PART III · ACCESS AND RULES

CHAPTER FIVE

Roles: Who May Do What, and for How Long

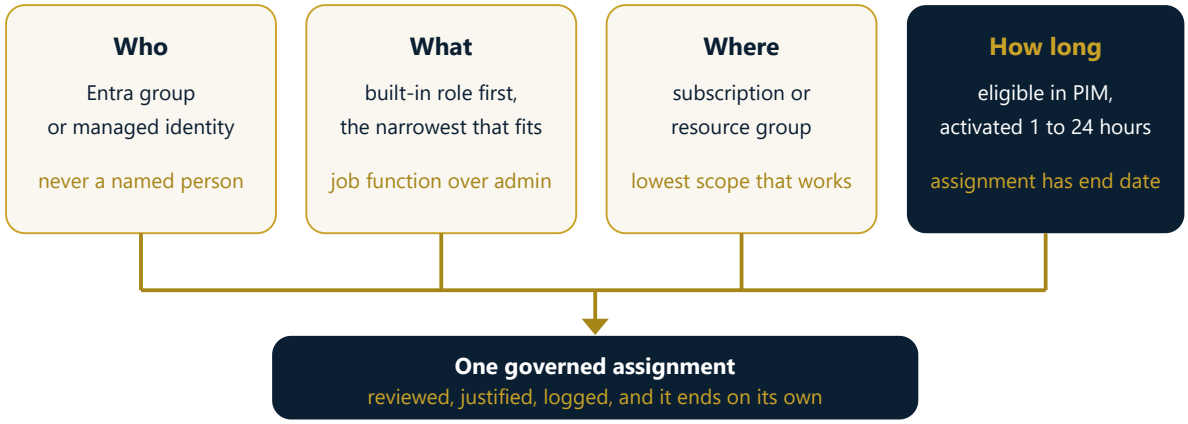


Access in most estates has a start date and no end date. Someone needed to fix something once, received Owner, and still has it. The fix for that is simple to say: give access to groups, not people; give the smallest role at the lowest scope; and make every powerful role temporary, activated only when needed and expiring on its own.

Azure role-based access control (Azure RBAC) answers the question "who may do what, where". Every role assignment has three parts: a security principal (a user, group, service principal or managed identity), a role definition (a set of permitted actions), and a scope (management group, subscription, resource group or resource). Assignments at a parent scope are inherited by every child. GDCD adds a fourth part that RBAC on its own leaves open: **for how long**.

FIGURE 5.1

The four questions of every role assignment



RBAC answers three questions. The contract needs four. Privileged Identity Management supplies the fourth: access that is eligible, activated for a bounded time, and recorded with a reason.

Built-in roles first, and know which ones are privileged

Azure has hundreds of built-in roles. Microsoft groups them into two kinds. Job function roles manage specific resource types, for example Virtual Machine Contributor or Storage Blob Data Reader. Privileged administrator roles grant broad management or the ability to grant access to others. Microsoft lists these as privileged:

Role	Why it is privileged	Governed default
Owner	Full management of all resources, and can assign roles	Eligible only, small platform group, approval required in prod
Contributor	Full management of all resources, cannot assign roles	Eligible for workload teams on their own subscriptions
User Access Administrator	Manages access, can grant itself or others Owner	Eligible, platform identity team only
Role Based Access Control Administrator	Manages access, can grant Owner unless constrained	Use with a condition that limits which roles it may assign
Reservations Administrator	Manages all reservations in the tenant	FinOps team only, eligible

Prefer a job function role whenever one exists. A team that needs to read blobs gets Storage Blob Data Reader, not Contributor and certainly not Owner. When you must delegate the ability to assign roles, Microsoft recommends adding a condition to the Role Based Access Control Administrator assignment so the delegate can only grant the specific roles you allow, for example only data reader roles, and never Owner. Microsoft also recommends a

maximum of three subscription owners, a check that Microsoft Defender for Cloud reports on. Create custom roles only when no built-in role fits, list their actions explicitly without wildcards, and reference roles by their ID in automation, because names can change.

Groups, not people

Assign roles to Microsoft Entra groups, never directly to users. It keeps the number of assignments small, which matters because Azure limits role assignments per subscription and per management group. More importantly, it moves the question "should this person have access?" to group membership, where joiners, movers and leavers processes and access reviews can answer it. A naming pattern helps everyone read the model, for example *grp-claims-prod-contributor*: workload, environment, role.

Just in time with Privileged Identity Management

Microsoft Entra Privileged Identity Management (PIM) turns standing access into eligible access. An eligible member holds the right to activate a role, not the role itself. To use it, they activate for a bounded time, and PIM can require, per role and per scope:

- An activation duration between 1 and 24 hours, set as a maximum by the administrator.
- Multifactor authentication, or a Conditional Access authentication context that demands a specific strength or a compliant device.
- A business justification, and optionally a ticket number.
- Approval from designated approvers. There are no default approvers, and Microsoft recommends at least two. The first approver to respond decides, so two approvers give availability, not dual control; where a change needs two people, enforce that in the pipeline or change process.
- An end date on the eligible assignment itself, so even eligibility expires and must be renewed.

PIM settings are defined per role and per resource, and settings at a subscription are not inherited by resource groups below it. Configure them deliberately at each scope where you grant access, from code where possible. PIM for Groups lets you make membership of a privileged group eligible, which is how a vended subscription can grant a workload team Contributor without granting it permanently. PIM requires Microsoft Entra ID P2 or Microsoft Entra ID Governance licenses for the users who hold eligible assignments and for approvers; confirm coverage before you design around it.

FIGURE 5.2

A role matrix by persona and environment (summary; full version in the appendix)

Persona	Development	Test	Production
Workload developer	Contributor, eligible, no approval	Reader standing; Contributor eligible	Reader standing; no write
Workload lead	Contributor, eligible	Contributor, eligible	Contributor, eligible, approval required
Platform engineer	Owner eligible at the Landing zones management group; separate eligible roles at the Platform management group or its subscriptions, least privilege; approval required, at least two approvers configured		
Deployment pipeline	Contributor on its own subscription	Contributor on its own subscription	Contributor on its own subscription, separate identity
Security operations	Security Reader standing; Security Admin eligible		

Production write access is never standing for a person. In production, people read; pipelines write. Humans write only through an approved, time-bound activation.

Workload identities: no shared keys

Pipelines and applications need identities too, and they cannot hold eligible roles in PIM, because they cannot complete an activation. Their role assignments are always active, which is why the scope must be narrow and the credential must not leak. The contract for workload identities:

- **Managed identities** for Azure resources talking to other Azure services. No secret exists to steal.
- **Workload identity federation** for pipelines in GitHub Actions or Azure DevOps. The pipeline exchanges its own token for an Entra token; no client secret is stored.
- **One identity per pipeline per environment.** The production pipeline's identity cannot deploy to development, and the reverse.
- **An inventory with owners.** Every service principal and managed identity has a named owner and a purpose; any remaining secret or certificate has a monitored expiry date; identities with no owner or no recent sign-in are reviewed and removed.

- **Entra ID authentication over keys.** Where a service supports Entra ID authentication, turn off key-based or local authentication, and enforce that with policy (Chapter Six).

Reviews and evidence

Access that is not reviewed drifts back to standing. Run access reviews on privileged groups and on eligible assignments, quarterly for production at a minimum, and let owners confirm or remove. Then check the result independently. Azure Resource Graph exposes role assignments, so a scheduled query can find what the contract forbids, for example a privileged role assigned directly to a user:

```
authorizationresources
| where type =~ "microsoft.authorization/roleassignments"
| extend roleDefinitionId = tostring(properties.roleDefinitionId),
         principalType = tostring(properties.principalType),
         scope = tostring(properties.scope)
| extend roleId = tostring(split(roleDefinitionId, "/")[-1])
| where roleId in~ (
    "8e3af657-a8ff-443c-a75c-2fe8c4bcb635",   // Owner
    "b24988ac-6180-42a0-ab88-20f7382dd24c",   // Contributor
    "18d7d88d-d35e-4fb5-a5c3-7773c20a72d9",   // User Access Administrator
    "f58310d9-a9f6-439a-9e8d-f62e7b41a168")   // Role Based Access Control Administrator
| where principalType == "User"
| project scope, roleDefinitionId, principalId = tostring(properties.principalId)
```

Active assignments appear in this table, including activated PIM roles while they are active. An empty result is one control result, not proof on its own: keep it with the query, the scope and the timestamp, next to the group membership and PIM reviews that cover access granted through groups. A non-empty result is a finding with a scope and a principal, ready for the owner to fix.

Two more populations need the same discipline. Guests from partner organizations should arrive through entitlement management access packages in Microsoft Entra ID Governance, which carry an approver and an expiration, and stay covered by recurring access reviews. Service principals that hold privileged roles can be included in PIM access reviews too, which needs Workload Identities Premium licenses on top of P2 or ID Governance, so a forgotten automation account is found by the same process as a forgotten person.

THE STANDING-ACCESS CLEANUP

In a typical first review, most direct user assignments turn out to be leftovers from projects that ended. The platform team does not remove them in one sweep. It publishes the list, converts the ones still needed into group-based eligible assignments, gives owners a fixed window, and then removes the rest. The Resource Graph query above becomes the control that keeps the number at zero.

> *Access should behave like a hotel key, not a house key: issued for a stay, logged at the door, and dead at checkout.*

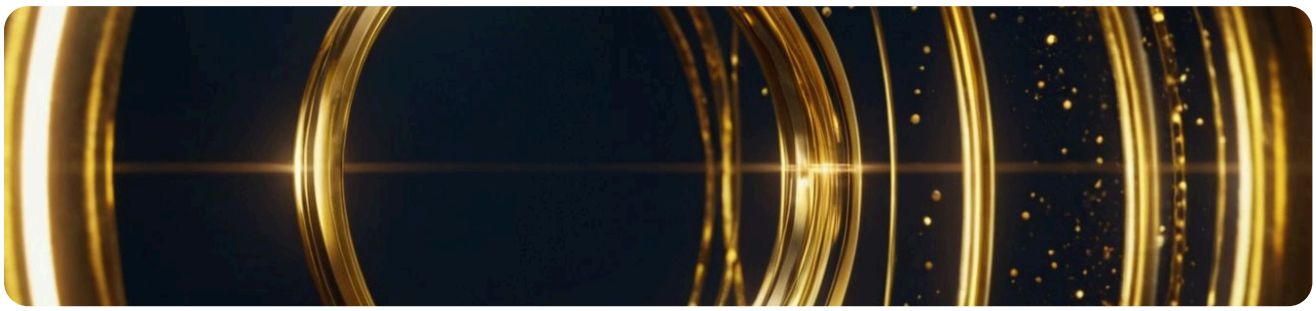
CONTRACT CHECKLIST

- All human role assignments are to groups, not users.
- No person holds standing Owner, Contributor or User Access Administrator in production.
- PIM requires MFA and justification for every privileged role, and approval for production.
- No subscription has more than three owners.
- Pipelines use workload identity federation or managed identities, one identity per environment, with no stored secrets.
- Access reviews run at least quarterly on privileged groups, and a scheduled query reports direct user assignments.

See it live: the roles map at hub.jcardena.com/azure/governance-roles.html

CHAPTER SIX

Policy Is the Contract That Runs

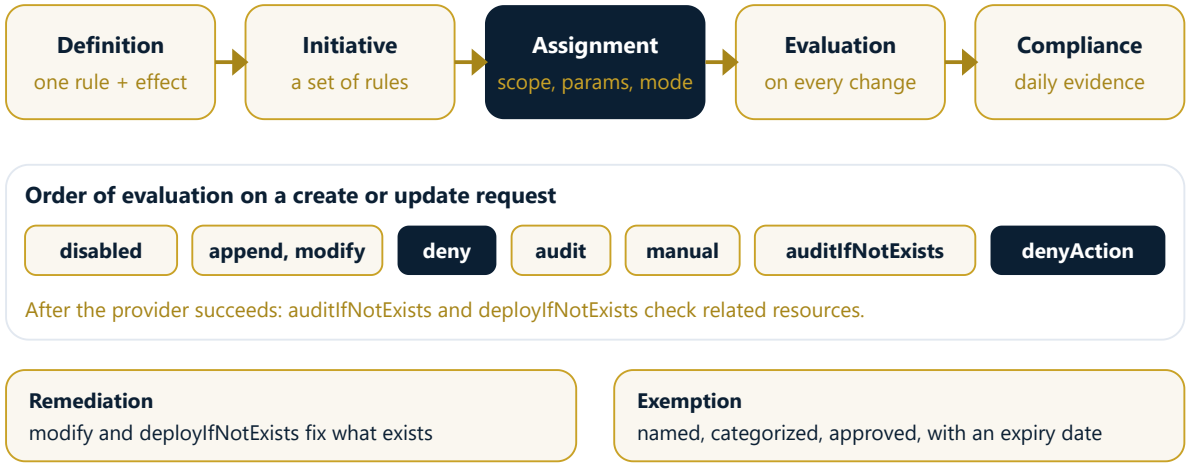


A standard written in a document is checked when someone remembers to check it. A standard written as Azure Policy is checked on every create and every update, by the platform, and reported every day. That is the difference between a wish and a contract. Policy is where the guarantees from Chapter One stop being promises and start being enforced.

Azure Policy evaluates resources against rules and reports or acts on what it finds. Each policy definition has one rule and one effect. Definitions are grouped into initiatives, also called policy sets, and both are assigned to a scope: a management group, a subscription or a resource group. Assignments inherit down the tree from Chapter Three, so one assignment on the Landing zones group governs every workload subscription beneath it.

FIGURE 6.1

The policy lifecycle: from rule to evidence



Assignment is where the contract is signed. The definition says what is checked. The assignment says where, with which parameters, in which enforcement mode, and with which identity if it must fix things.

The effects, in plain words

Azure Policy supports these effects: addToNetworkGroup, append, audit, auditIfNotExists, deny, denyAction, deployIfNotExists, disabled, manual, modify and mutate. Five carry most enterprise governance, and each is a different kind of contract clause.

Effect	What it does	Contract clause
Audit	Allows the request and marks the resource non-compliant. Judges the resource on its own properties.	"We will know." Observe before you enforce.
AuditIfNotExists	Marks a resource non-compliant when a related resource is missing, for example a VM without a required extension.	"We will know if the companion is missing."
Modify	Adds, updates or removes properties or tags on create or update, and can remediate existing resources.	"We will correct it for you." Typical for tags.
DeployIfNotExists	Deploys a related resource when it is missing, for example diagnostic settings that send logs to the central workspace.	"We will add what is required."
Deny	Rejects the create or update request before it reaches the resource provider.	"This will not happen."

Two details matter in practice. Modify and DeployIfNotExists change resources, so their assignments need a managed identity with the right roles to do the work, and existing resources are fixed through remediation tasks. And when several assignments apply to the same resource, each is evaluated independently and the net result is cumulative most restrictive: if any assignment denies, the request is denied.

A starter set that follows the outcomes

Every policy in the set should trace back to a guarantee. The landing zone accelerators assign a sensible baseline, and we add the rules our outcomes require. A typical starter set, by scope:

Guarantee	Policy (built-in where one exists)	Effect	Scope
Resources deploy only to approved regions	Allowed locations	Deny	Landing zones
Every resource has an owner and cost center	Inherit a tag from the subscription if missing	Modify	Intermediate root
No public PaaS endpoints for corp workloads	Public network access should be disabled for PaaS services (landing zone initiative)	Deny	Corp
Every subscription sends logs to one place	Diagnostic settings to the central workspace	DeployIfNotExists	Intermediate root
Security baseline is measured	Microsoft cloud security benchmark	Audit, AuditIfNotExists	Intermediate root

The full starter set, with effects for sandbox and landing zones, is in the appendix. Network and security policies are covered with their layers in Chapters Seven and Eight.

Note what Allowed locations promises and what it does not. It controls the location of the resources it covers. It does not, on its own, prove where data travels: global services, geo-replication, backup storage and support processes each have their own settings. A data residency guarantee needs those service settings as well, written into the same contract.

Audit first, then deny, and the trap in "audit in dev"

A common rule of thumb is "audit in dev, deny in prod". We use a different one, because Microsoft's landing zone guidance warns against giving development, test and production different guardrails: teams build against the looser rules, then fail at the last promotion step. Our rule is **audit first, deny once clean, and the same guardrails in every environment**. The difference between audit and deny is a stage in time, not a difference between environments. Sandbox is the only place with a deliberately looser set.

One distinction keeps the rule honest. Guardrails are the rules about location, network exposure, identity and security configuration, and they are identical in dev, test and prod. Cost and scale parameters, such as the allowed SKU list, auto-shutdown or the depth of resource logging, may differ by environment (Chapter Ten); set them as parameters at subscription scope during vending. Production may add obligations, such as backup. No environment removes a guardrail.

Azure Policy gives you the tools to stage that rollout safely:

- **Enforcement mode.** An assignment with enforcementMode set to DoNotEnforce evaluates and reports compliance without enforcing the effect, so you can see the impact of a Deny before it blocks anyone.
- **Resource selectors.** Limit a new assignment to certain regions or resource types first, then widen it in steps.
- **Overrides.** Change the effect of selected definitions inside an initiative, for example from audit to deny, without editing the definitions.
- **Non-compliance messages.** A Deny should tell the engineer why, and where to ask. A custom message is part of the contract.

```
targetScope = 'managementGroup'

resource allowedLocations 'Microsoft.Authorization/policyAssignments@2024-04-01' = {
  name: 'allowed-locations'
  properties: {
    displayName: 'Resources only in approved regions'
    policyDefinitionId: '/providers/Microsoft.Authorization/policyDefinitions/e56962a6-
4747-49cd-b67b-bf8b01975c4c'
    enforcementMode: 'DoNotEnforce'   // stage 1: report only; switch to 'Default' once
clean
    parameters: {
      listOfAllowedLocations: { value: [ 'westeurope', 'northeurope' ] }
    }
    nonComplianceMessages: [
      { message: 'Region not approved. See the platform guide or contact the platform
team.' }
    ]
  }
}
```

Where an environment truly needs an extra obligation, such as backup only in production, Microsoft suggests assigning it at subscription scope during vending, or auditing at management group scope and letting the workload team implement it. Keep security guardrails at management group scope. Subscription-scoped assignments can be changed by

people with elevated rights on that subscription, which is also why permissions to write policy assignments, definitions and exemptions should never be standing; put them behind PIM.

Exemptions with an expiry date

Every real estate has exceptions. A contract without a way to handle them gets bypassed. Azure Policy exemptions are the sanctioned path, and they carry exactly the fields a governance contract needs: which assignment, which definitions within it, a category, metadata for who asked and who approved, and an expiry.

```
{
  "properties": {
    "displayName": "Claims legacy API: public endpoint until migration",
    "policyAssignmentId": "/providers/Microsoft.Management/managementGroups/org-
corp/providers/Microsoft.Authorization/policyAssignments/deny-public-endpoints",
    "exemptionCategory": "Waiver",
    "expiresOn": "2027-03-31T23:59:00.0000000Z",
    "metadata": {
      "requestedBy": "claims workload team",
      "approvedBy": "platform security lead",
      "ticketRef": "CHG-0000"
    }
  }
}
```

There are two categories. **Waiver** means the non-compliance is temporarily accepted. **Mitigated** means the intent of the policy is met another way. When expiresOn passes, Azure does not delete the exemption; it keeps the object for record-keeping and simply stops honoring it, so the resource becomes non-compliant again and, under Deny, its next update is blocked. Creating an exemption requires its own permission on the target assignment, beyond normal write access, and built-in roles such as Resource Policy Contributor and Security Admin can manage them.

OUR EXEMPTION CONTRACT

- Every exemption has an expiry date. We set a maximum of one quarter for waivers, renewable once with a fresh approval.
- Waivers need a migration plan. Mitigated exemptions name the compensating control, its owner and the evidence that it runs, for example a web application firewall in front of a public endpoint plus an alert on its logs.
- All exemptions and their compensating controls sit in one register, generated from the exemption metadata, that an auditor can read without asking.
- Requester, approver and ticket live in the metadata, not in someone's inbox.
- A monthly report lists exemptions expiring in the next 30 days and expired ones still on file.

Exempt is not excluded. An exemption keeps the resource visible with a compliance state of Exempt, and Azure records what its state would be without the exemption. Excluding a scope with notScopes removes it from evaluation entirely. Use exemptions for exceptions, and keep notScopes for structural reasons you can defend.

WHEN THE CONTRACT DECIDES

A workload team asks for a public endpoint on a storage account in production because a partner integration needs it this week. Under a document-based standard, the answer depends on who is asked. Under policy, the Deny on Corp decides, and the path is clear: request a Waiver, state the plan to move to private connectivity, get approval, receive an exemption that expires at the end of the quarter. The engineer moves on in a day. The exception is visible, owned and temporary.

The model proposes. The contract decides. Here the model is an engineer's request, and the exemption, with its date, is how the contract says "yes, for now".

CONTRACT CHECKLIST

- Every assigned policy traces to a written guarantee and an owner.
- Guardrails are assigned at management group scope and are the same for dev, test and prod; only Sandbox differs.
- New Deny assignments run in DoNotEnforce or audit first, with a date to switch to enforcement.
- Modify and DeployIfNotExists assignments have a managed identity with least-privilege roles, and remediation runs on existing resources.
- Every exemption has a category, approver metadata and an expiry date, and expiring ones are reported monthly.
- Rights to change policy assignments, definitions and exemptions are eligible through PIM, never standing.

See it live: the policy map at hub.jcardena.com/azure/governance-policy.html

PART IV · GUARANTEE LAYERS

CHAPTER SEVEN

Network: Private by Default



Here is the point in plain words: in a governed estate, no data service is reachable from the internet unless someone wrote down why, and the platform checks that promise every time a resource is created. Private is the default. Public is the exception, and an exception has an owner, a reason and an end date.

Parts II and III gave the estate its structure and its rules. Chapter Three placed a Connectivity management group under Platform. Chapter Four gave every workload its own subscription per environment. Chapter Six made policy the contract that runs. This chapter is where those pieces meet the wire. The network is the first guarantee layer because it decides what can even reach a resource before identity, encryption or monitoring get a chance to help.

The guarantee we write is short: **data services in every workload environment accept traffic only over private paths; all outbound traffic leaves through a point we inspect and log; name resolution works the same from every spoke and from the office.** Every line holds, or the guarantee is a wish.

Choose the shape first: hub and spoke, or Virtual WAN

Microsoft's Cloud Adoption Framework describes two network topologies for a landing zone. Both put shared network services in one central place and connect workload networks to it. They differ in who runs the routing.

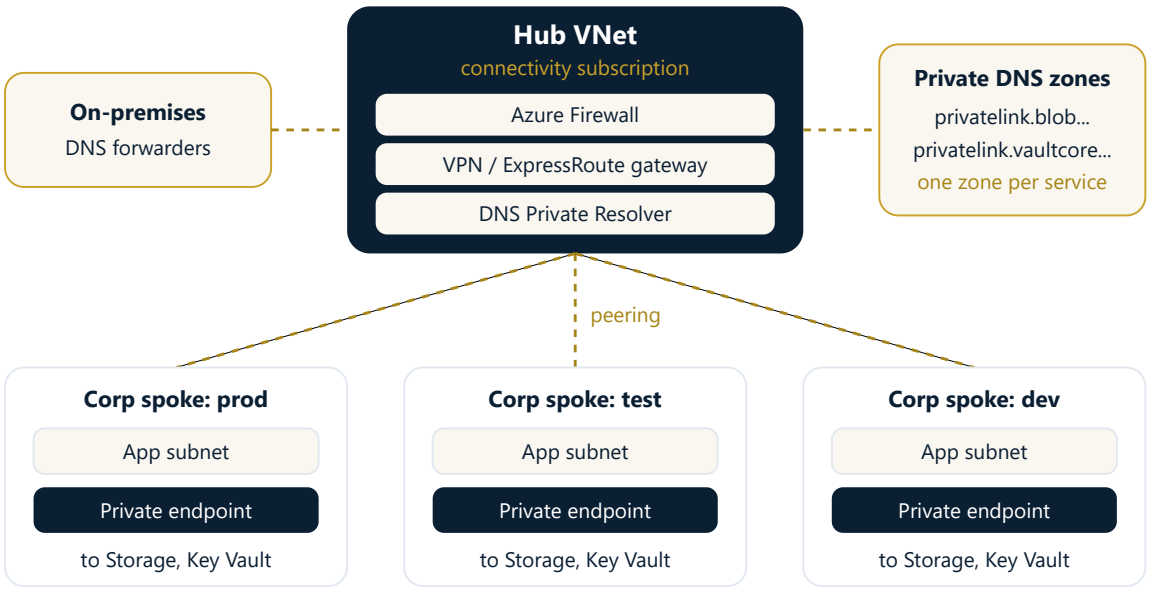
- **Traditional hub and spoke.** You build the hub virtual network yourself, peer each spoke to it, and manage routing and security. The framework suggests it when you deploy in one or a few regions, have a modest number of branch sites, and want full, manual control of routing.

- **Azure Virtual WAN.** A Microsoft-managed service runs the hubs and much of the routing. The framework suggests it for global connectivity across many regions and many on-premises locations, large SD-WAN branch networks, or when you need transitive routing between VPN and ExpressRoute.

Choose by those criteria, not by habit. Hub and spoke is often the simpler start in one or a few regions, because it is easier to reason about and to explain to an auditor; Virtual WAN earns its place when managed, global routing is the requirement. The contract is the same either way. The pattern outlives the product choice: one hub per region, owned by the platform team, sitting in the connectivity subscription; spokes owned by workload teams, sitting in their own landing zone subscriptions; no spoke talks to another spoke except through the hub.

FIGURE 7.1

Hub and spoke with private endpoints and central DNS



The hub owns the shared doors; the spokes own their workloads. Every spoke reaches data services through a private endpoint in its own subnet. The DNS records for those endpoints live in central private DNS zones that only the platform team manages, so every spoke and the office resolve the same name to the same private address.

Private endpoints: the data door

A private endpoint is a network interface in your virtual network that gets a private IP address and connects to one platform service, such as a storage account, a key vault or a database. Traffic to that service then stays on private paths. Pair it with turning off public

network access on the service itself, and the service refuses calls from the internet no matter what anyone types.

That last step matters. Microsoft's documentation is explicit that DNS resolution and access control are independent. The public name of a resource still resolves from anywhere; what stops an outside caller is the service's own public network access setting or firewall. A private endpoint without that setting is a second door, not a locked first one.

DNS is where private endpoints quietly fail

Applications keep using the normal service name, for example the blob endpoint of a storage account. Azure's public DNS answers with an alias into a `privatelink` zone, and your private DNS zone answers that alias with the private IP. If the private zone is missing, linked to the wrong network, or duplicated in a spoke, the application either reaches the public endpoint (and gets refused) or fails to resolve at all. The symptom looks like an application bug. The cause is a broken contract at the DNS door.

The Cloud Adoption Framework pattern for doing this at scale is the one we run:

1. The platform team creates the `privatelink` private DNS zones once, in the connectivity subscription, using Microsoft's recommended zone names (for example `privatelink.blob.core.windows.net` for blob storage and `privatelink.vaultcore.azure.net` for Key Vault). Use those exact names: the public DNS alias for each service points into the zone of that name, so a zone with any other name never answers. Private DNS autoregistration is a different feature; it creates records for virtual machines only, not for private endpoints.
2. The zones are linked to the hub, and every network resolves through a resolver in the hub, such as Azure DNS Private Resolver. On-premises DNS servers conditionally forward the public service zones to the resolver's inbound endpoint, for example `blob.core.windows.net`, not the `privatelink` name, as Microsoft's private endpoint DNS guidance specifies.
3. A **Deny** policy stops workload teams from creating their own zones with the `privatelink` prefix.
4. A **DeployIfNotExists** policy watches for new private endpoints and adds a DNS zone group that writes the record into the central zone. Its managed identity needs the Private DNS Zone Contributor role on the zones. When the endpoint is deleted, the record goes with it.

The landing zone reference implementation ships an initiative for this, titled "Configure Azure PaaS services to use private DNS zones". Prefer built-in or reference definitions over hand-written ones, and keep them current.

The deny rule, written once

Chapter Six set the convention: the same guardrails in dev, test and prod, audit first and deny once the estate is clean. For network exposure the rule per service is small. This one refuses any storage account in scope that leaves public network access on:

```
{
  "mode": "All",
  "policyRule": {
    "if": {
      "allOf": [
        { "field": "type", "equals": "Microsoft.Storage/storageAccounts" },
        { "field": "Microsoft.Storage/storageAccounts/publicNetworkAccess",
          "notEquals": "Disabled" }
      ]
    },
    "then": { "effect": "[parameters('effect')]" }
  },
  "parameters": {
    "effect": { "type": "String", "allowedValues": ["Audit", "Deny", "Disabled"],
                "defaultValue": "Audit" }
  }
}
```

The definition is assigned once, at the Landing zones management group, with the effect parameter set to `Audit`. When the compliance view is clean, the same assignment switches to `Deny` on a date agreed in advance. Dev, test and prod get the same rule at the same strength, and the switch date applies to the whole landing zone scope at once.

Public front doors, segmentation and DDoS

Private by default applies to data services. Online landing zones still publish applications to the internet, and the contract decides how: through one governed front door, Azure Front Door or Application Gateway with a Web Application Firewall (WAF) policy, while the databases and storage behind it stay private. The built-in policies *Azure Web Application Firewall should be enabled for Azure Front Door entry-points* and *Web Application Firewall (WAF) should be enabled for Application Gateway* make that rule checkable.

Azure DDoS Protection works at network layers 3 and 4; the WAF covers layer 7. DDoS Network Protection is a plan linked to virtual networks, and one plan can serve virtual networks across subscriptions in the tenant; DDoS IP Protection is the per-IP alternative. Without either, the default infrastructure protection still applies. Decide per archetype, typically Online and any spoke with public IP addresses, record the cost with Chapter Ten, and audit coverage with *Virtual networks should be protected by Azure DDoS Protection*.

Inside the network, every workload subnet gets a network security group and a route table that sends internet-bound traffic to the hub firewall. Application security groups let rules name roles, such as web, app and data, instead of IP addresses. *Subnets should be associated with a Network Security Group* audits the first half; the route table is set by vending with the spoke.

Egress: one way out, and it writes a log

Private inbound is half the guarantee. The other half is outbound. Spokes send internet-bound traffic to the firewall in the hub through a route table, and the firewall's diagnostic logs flow to the central workspace you will meet in Chapter Nine. When a workload needs a new outbound destination, it is a firewall rule change through the pipeline in Chapter Eleven, not a public IP attached in a hurry.

FIGURE 7.2

The network contract, staged in time

Rule	Stage 1: audit, days 1 to 60	Stage 2: enforced	Applies to
Public network access on data services	Audit	Deny	Dev, test and prod
Private DNS zones created in spokes	Audit or DoNotEnforce	Deny	Dev, test and prod
Private endpoint DNS records	DeployIfNotExists into central zones		Dev, test and prod
Public IP on virtual machine NICs	Audit	Deny	Dev, test and prod
Internet egress	Through the hub firewall, logged centrally		Every spoke

Same rules in every environment, staged in time. Audit is a first stage, not a dev setting, so nothing is built in dev that prod would refuse. Sandbox keeps its own looser set and has no path to the hub.

A COMPOSITE EXAMPLE: THE DATA TEAM'S QUICK FIX

A team cannot reach its new storage account from a build agent and asks for public access "for one week". In an ungoverned estate, that week becomes a year. Here the request is a policy exemption with an owner, a reason and an expiry date (Chapter Six). The usual root cause turns out to be a private endpoint created without a DNS record, which the DeployIfNotExists policy now fixes on its own.

Watch for: virtual networks that use custom DNS servers which do not forward to the hub resolver; private endpoints created with "Integrate with private DNS zone" set to yes, which tries to create a zone in the spoke and is refused; and services that are reachable privately but still have public network access enabled.

> *The network is not secure because it is private. It is secure because the platform refuses to let it be anything else, and writes down every exception.*

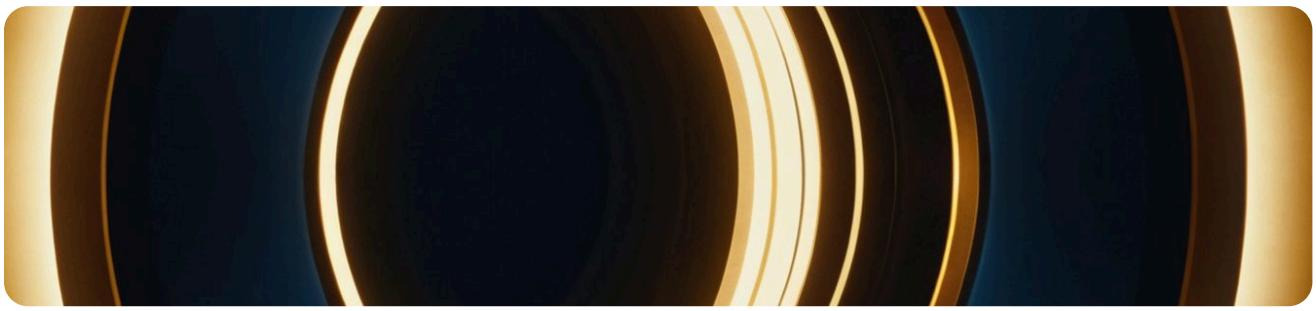
CONTRACT CHECKLIST

- Topology decided and written down: hub and spoke or Virtual WAN, with the reason.
- Central `privatelink` zones exist in the connectivity subscription, with recommended names.
- Deny policy for public network access is assigned at the Landing zones management group, so dev, test and prod get the same rule.
- DeployIfNotExists policy registers private endpoint DNS records automatically.
- All internet egress passes through the hub firewall and its logs reach the central workspace.
- Public applications publish only through a WAF front door; DDoS protection is decided per archetype; every workload subnet has a network security group, except special subnets where Azure documents otherwise, such as GatewaySubnet and AzureFirewallSubnet.

See it live: the landing zone map at hub.jcardena.com/azure/governance-landing-zone.html

CHAPTER EIGHT

Security Posture You Can Watch



The plain point first: security you cannot see is security you are only hoping for. A governed estate measures its own posture every day, shows that measure to the people who own the risk, and turns the most common fixes into rules so the same finding never comes back.

Chapter Seven closed the network doors. This chapter covers what sits behind them: the posture of every resource, the secrets that applications use, the identities that use them, the few resources that must never be deleted by accident, and the promise that data can come back after a bad day. Each of these is a guarantee. Each needs an enforcer and a piece of evidence, or it is not a guarantee at all.

Defender for Cloud: the posture meter

Microsoft Defender for Cloud continuously assesses Azure resources against a security standard. When it is turned on for a subscription, the Microsoft cloud security benchmark (MCSB) is applied by default, and every finding becomes a recommendation with the affected resources listed.

Two posture plans exist, and the difference matters for budgeting:

- **Foundational CSPM** is free. It gives asset inventory, the MCSB recommendations, secure score, remediation tools and workflow automation.
- **Defender CSPM** is a paid plan. It adds attack path analysis, the cloud security explorer, risk prioritization, agentless vulnerability scanning, data security posture and governance rules that assign findings to owners with due dates.

Separate workload protection plans (for servers, storage, databases, containers, Key Vault and others) add threat detection for those resource types. The pattern we use is simple: Foundational CSPM everywhere from day one, Defender CSPM and the workload plans on

production and on anything that holds regulated data, decided per resource type and written into the cost model from Chapter Ten. Prices and billable resources change, so check the current Defender for Cloud pricing page before you commit.

Secure score: one number, read correctly

Secure score rolls the findings into a single percentage per subscription, per management group and for the whole estate. Higher means less identified risk. Two details keep it honest:

- Each security control earns points in proportion to its healthy resources: its maximum score is shared across the resources it assesses, and each healthy resource adds its share. The full points come only when *all* its recommendations are fixed for *all* its resources, so fixing one recommendation on a resource that still fails another in the same control moves little.
- Microsoft now offers two models. The classic secure score remains in the Azure portal. A risk-based Cloud Secure Score, which weighs asset risk factors and criticality, lives in the Microsoft Defender portal. They are calculated differently and produce different values, so pick one for your scorecard and say which.

Because the score is data, it can be queried. This Azure Resource Graph query, taken from Microsoft's documentation pattern, returns the score per subscription so the operating review in Chapter Twelve can trend it:

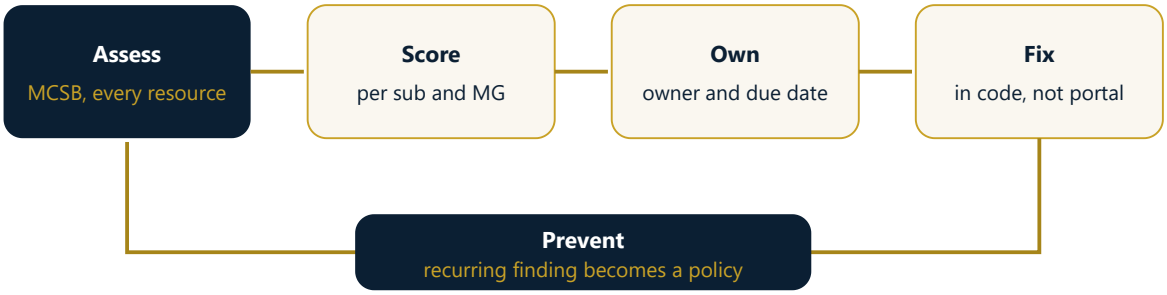
```
SecurityResources
| where type == 'microsoft.security/securescores'
| extend current = properties.score.current, max = todouble(properties.score.max)
| project subscriptionId, current, max, percentage = ((current / max) * 100)
```

Regulatory compliance: the auditor's view

Auditors ask about standards, not scores. The regulatory compliance dashboard in Defender for Cloud maps assessments to the controls of standards such as ISO/IEC 27001:2022, PCI DSS v4.0.1, NIST SP 800-53 and the EU Digital Operational Resilience Act (DORA). MCSB is assigned by default. Adding other standards needs a paid Defender for Cloud plan (Defender for Servers Plan 1 and Defender for APIs Plan 1 do not qualify) and Owner or Resource Policy Contributor rights. Underneath, each standard is an Azure Policy initiative, so assign it at the highest scope it applies to, usually the intermediate root. Two cautions: controls that cannot be assessed automatically appear greyed out and still need manual evidence, and a green dashboard is evidence for an audit, not a certification.

FIGURE 8.1

The posture loop: measure, own, fix, prevent



Each turn of the loop should leave one fewer finding to fix by hand.

A score is only useful inside a loop. Defender for Cloud offers Enforce and Deny actions on many recommendations; the platform team uses them, through policy (Chapter Six), to turn a repeated fix into a standing rule.

Key Vault: secrets with an owner and a boundary

Secrets, keys and certificates live in Azure Key Vault, never in code, pipeline variables or app settings. Four practices carry most of the weight:

- **One vault per application per environment.** This is Microsoft's own recommendation. A shared vault is a shared blast radius, and role assignments on individual secrets are discouraged except in narrow cases.
- **Azure RBAC as the permission model.** Microsoft states that, starting with Key Vault API version 2026-02-01, Azure RBAC is the default access model for newly created vaults; legacy access policies remain for older vaults. Use built-in data roles: **Key Vault Secrets User** for an application that reads secrets, **Key Vault Secrets Officer** for the team that manages them, **Key Vault Crypto User** for cryptographic operations, **Key Vault Reader** for metadata only. Note that **Key Vault Contributor** manages the vault itself and grants no data access.
- **No public door.** Public network access disabled, private endpoint as in Chapter Seven, and purge protection turned on for production vaults, so a deleted vault or secret stays recoverable through its soft-delete retention period and nobody can purge it early.
- **Keys you control where required.** Azure services encrypt data at rest by default with Microsoft-managed keys. Where the data classification or a regulator demands it, use customer-managed keys held in Key Vault for the services that support them, with a rotation policy on each key. Treat the key as a production dependency: if it is disabled,

deleted or its access is revoked, the services using it lose access to their data. Keep purge protection on the vaults that hold such keys, keep key administrators separate from data owners, and include key recovery in the recovery-region plan.

Applications reach the vault with a **managed identity**, not a stored credential. A system-assigned identity lives and dies with its resource; a user-assigned identity can be shared by a few resources that genuinely form one workload. Either way, there is no password to rotate and nothing to leak in a log.

Resource locks: a seatbelt, not a policy

A resource lock overrides every user's permissions on the control plane. **CanNotDelete** allows changes but blocks deletion. **ReadOnly** blocks changes and deletion. In the portal the same two levels are called Delete and Read-only. Locks inherit down from a subscription or resource group, and only roles with `Microsoft.Authorization/locks/*`, such as Owner and User Access Administrator, can create or remove them.

Use CanNotDelete on the small set of things whose loss would stop the estate: the hub network, the central Log Analytics workspace, production key vaults, backup vaults, the private DNS zones. Be careful with ReadOnly. Microsoft documents side effects: on a storage account it blocks listing keys; on a resource group it can stop virtual machines from starting and block App Service scaling. Locks also apply only to the control plane. Deleting a blob, a secret or the rows of a database still works under any lock, so protect data with soft delete, versioning and backup instead.

```
param location string = resourceGroup().location
param kvName string

resource kv 'Microsoft.KeyVault/vaults@2023-07-01' = {
  name: kvName
  location: location
  properties: {
    tenantId: subscription().tenantId
    sku: { family: 'A', name: 'standard' }
    enableRbacAuthorization: true
    enablePurgeProtection: true
    publicNetworkAccess: 'Disabled'
  }
}

resource kvLock 'Microsoft.Authorization/locks@2016-09-01' = {
  name: 'kv-cannot-delete'
  scope: kv
  properties: {
    level: 'CanNotDelete'
    notes: 'Production secrets vault. Removal needs a change record.'
  }
}
```

Patching has an owner

Vulnerability findings close only when someone patches. Azure Update Manager assesses and installs updates for Azure virtual machines and Azure Arc-enabled servers; built-in policies turn on periodic assessment, and maintenance configurations set patch windows per environment. The RACI in Chapter Twelve names the owner: the workload team for its own machines, the platform team for platform machines.

Backup and recovery: a promise with numbers

"We have backups" is an assertion. The guarantee is a recovery point objective (how much data you may lose) and a recovery time objective (how long you may be down) per workload tier, agreed with the business owner, plus evidence that a restore was tested. We record both targets as tags on the production subscription, protect backup vaults with locks, and schedule a restore test each quarter whose result is reviewed in Chapter Twelve. Turn on soft delete and immutability on Recovery Services and Backup vaults, and for regulated data lock the immutability setting. Only locked immutability is irreversible, so only then can a compromised administrator not switch it off and erase the restore points. Azure Policy has built-in definitions that audit whether virtual machines are protected by Azure Backup, and others that configure it; audit at the Landing zones group and configure at production subscriptions during vending, as Chapter Six allows for production-only obligations.

Backup answers data loss. Resilience answers an outage, and it is decided per workload tier:

- **Zone redundancy.** In regions with availability zones, choose zone-redundant options for production data and compute where the service offers them, so the loss of one datacenter is not an outage.
- **A named recovery region.** Decide per tier whether the workload recovers in a second region, and how: Azure Site Recovery for virtual machines, geo-replication or geo-redundant storage for data services. Not every region has a paired region, so name the recovery region explicitly instead of assuming one.
- **The platform recovers too.** The hub, the DNS zones, the policy set and the vending pipeline must be redeployable from code into the recovery region. Chapter Eleven is what makes that possible.
- **Proof.** A failover test per tier, at least once a year, with the measured recovery time recorded against the RTO.

> **A COMPOSITE EXAMPLE: THE SCORE THAT BARELY MOVED**
>
> A platform team fixed dozens of findings and watched the secure score barely move. Every fix was correct, but the fixes were spread thinly: one recommendation here, another there, on resources that still failed other recommendations in the same controls. Few resources became fully healthy, so few points were added. The team switched to closing whole controls on whole sets of resources, starting with the highest maximum score, and moved the recurring fixes into deny policies. The score then rose steadily and stayed there.

> *Posture is not a project you finish. It is a meter you read, a loop you run, and a list of rules that grows every time the same mistake appears twice.*

CONTRACT CHECKLIST

- Defender for Cloud is on for every subscription; Defender CSPM and workload plans are decided per environment and recorded in the cost model.
- The scorecard names which secure score model it uses and trends it monthly.
- Every application has its own vault per environment, using Azure RBAC and a managed identity.
- CanNotDelete locks protect the hub, central logs, prod vaults, DNS zones and backup vaults.
- Each production workload has RPO and RTO targets and a dated, tested restore.
- Each production tier has a named recovery region, or a written decision not to have one, and a failover test on record.

See it live: the policy and posture page at hub.jcardena.com/azure/governance-policy.html

CHAPTER NINE

Visibility: One Place to See Everything



In plain words: every change, every resource log and every alert in the estate lands in one place that the platform team owns, and every alert that matters reaches a named person. If an event cannot be found later, for the purposes of governance it did not happen, and neither did the control that was supposed to catch it.

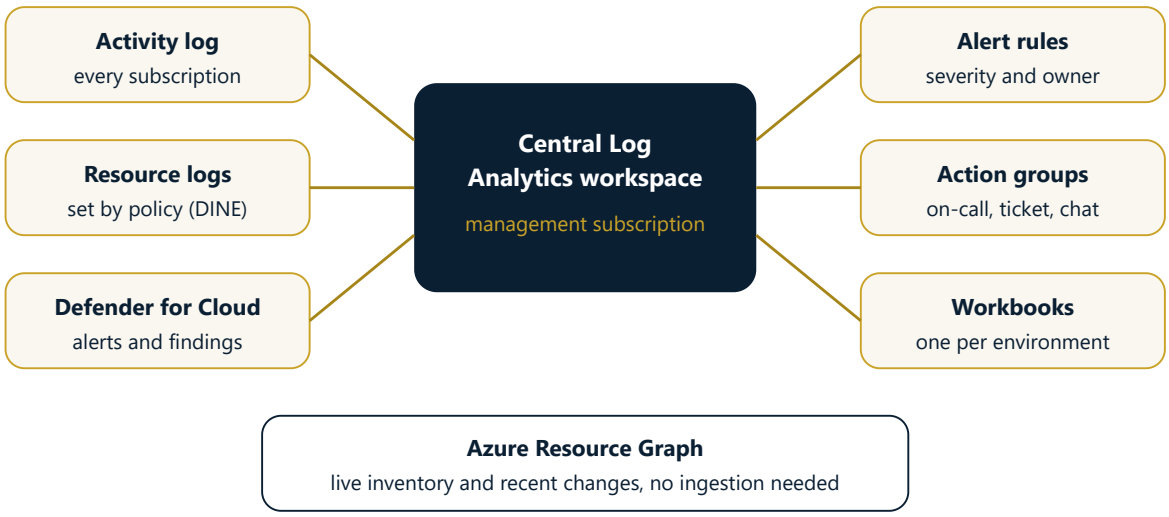
The earlier chapters wrote contracts for the tenant, the tree, the subscriptions, the roles, the policies, the network and the posture. Each contract has an evidence clause. This chapter is where that evidence is collected. In GDCD terms, visibility is the layer that turns assertions into records: who did what, where, when, and what the system decided.

The three signals and where they come from

- **The activity log** records control plane events per subscription: who created, changed or deleted a resource, role assignments, policy decisions. It is kept by the platform for a limited time, so we send it to the central workspace with a subscription-level diagnostic setting.
- **Resource logs** are produced by each service (Key Vault access, firewall decisions, database audits). They are off until a diagnostic setting turns them on, resource by resource. This is the gap most estates never close by hand.
- **Metrics and alerts**, from Azure Monitor and from Defender for Cloud (Chapter Eight), describe health and threats in near real time.

FIGURE 9.1

Signals in, decisions out



One workspace, many readers, with the exceptions below. Logs flow to one owned place; alerts flow out to named people. Resource Graph sits beside the workspace: it answers "what exists right now" across every subscription you can read, without collecting anything.

One workspace first, more only for a reason

Microsoft's workspace design guidance is direct: start with a single Log Analytics workspace and add more only when a criterion demands it. The main criteria it lists:

- **Tenants.** Most resources can only send data to a workspace in their own Microsoft Entra tenant, so a second tenant usually means a second workspace.
- **Data residency.** A workspace lives in one region; a legal requirement to keep data in a geography means a workspace there.
- **Security operations.** Teams that run Microsoft Sentinel often keep a dedicated security workspace, because Sentinel pricing applies to all data in a workspace where it is enabled. Others combine both to reach a commitment tier. Decide once, write down why.
- **Retention.** Retention can be set per table, so different retention for the same table from different resources is the only retention reason to split.

In the Azure landing zone layout, the workspace sits in the management subscription under the Platform branch from Chapter Three, owned by the platform team and protected by a CanNotDelete lock (Chapter Eight).

Diagnostic settings are a policy, not a ticket

Turning on resource logs by hand does not scale and does not survive the next new resource. Azure Policy ships built-in initiatives that do it for every supported resource type, one per destination and per log category group. The two category groups are `allLogs` and `audit`. For example, the initiative **"Enable allLogs category group resource logging for supported resources to Log Analytics"** deploys a diagnostic setting on each supported resource in scope.

Three details make it work. The effect is **DeployIfNotExists**, so it acts on new resources as they are created. Existing resources need a **remediation task**, one per policy in the initiative. And the assignment needs a **managed identity** with a role that can write diagnostic settings at the assignment scope. Assign the audit category group initiative once at the intermediate root, so platform subscriptions are covered as well as landing zones, and every subscription the vending process creates (Chapter Four) arrives with logging already on. Add the allLogs initiative at production subscriptions during vending.

The category group is a cost parameter, chosen per environment. Every environment sends `audit` through the intermediate root assignment; production also sends `allLogs` for data services and security-relevant resources through the assignment added at vending, which keeps ingestion cost in line with Chapter Ten.

Who may read what

Visibility must not become a new leak. By default, a workspace uses resource-context access: a person who can read a resource can read that resource's logs, without access to the whole workspace. Keep that default for workload teams. Grant workspace-wide access to the platform and security teams only, and use table-level access control where a table holds data a workload team should not see.

Evidence has a retention clause

Logs are evidence only for as long as you keep them. Analytics retention can be extended up to two years per table, and total retention, including low-cost long-term retention, up to 12 years; data past its analytics period is retrieved with search jobs. Set retention per table to match the record-keeping period your regulators and auditors require. Where evidence must be tamper-proof, also export the critical tables to a storage account with an immutability policy. If security operations run Microsoft Sentinel on the workspace, write down the split: the platform team owns the pipes and the retention, the security team owns the detections and the incidents.

Alerts that reach a person

An alert that reaches nobody is only a log entry. Every alert rule has three things: a severity, an action group that names where it goes (on-call rotation, ticket queue, chat channel), and an owner who decides whether it is still useful. We keep the list small. A handful of platform alerts cover the estate: a policy assignment or role assignment changed at management group scope, a lock removed, a deletion in production, a diagnostic setting removed, a high-severity Defender alert, and Azure Service Health and Resource Health alerts for the regions and services the estate uses, so a platform incident or planned maintenance reaches the on-call rotation before users call. Workload teams own their own service alerts in their spokes.

A starter query for the "deletions in production" alert, run against the central workspace:

```
AzureActivity
| where CategoryValue == "Administrative" and ActivityStatusValue == "Success"
| where OperationNameValue endswith "/DELETE"
| summarize deletes = count() by Caller, SubscriptionId, bin(TimeGenerated, 1h)
```

Resource Graph: the inventory that is always current

Azure Resource Graph answers questions about what exists across every subscription you can read, using a query language based on KQL. It needs nothing deployed and costs nothing to query, though queries are throttled. It also keeps the last 14 days of resource configuration changes, which is often enough to answer "what changed since Tuesday". We use it for the questions that a log workspace answers poorly: which resources have no owner tag, which storage accounts still allow public access, which subscriptions have no budget.

```
resources
| where isnull(tags['owner']) or tostring(tags['owner']) == ''
| summarize untagged = count() by subscriptionId, type
| order by untagged desc
```

Each environment gets one workbook built on these queries and on the workspace: compliance from policy, secure score from Chapter Eight, open alerts, untagged resources and spend from Chapter Ten. That workbook is the page the operating review in Chapter Twelve opens first.

A COMPOSITE EXAMPLE: THE MISSING SIX HOURS

An incident review needed to know who changed a production firewall rule. The activity log showed the change, but resource logs for the firewall had never been switched on, so nobody could see which traffic the old rule had allowed. The fix was not a new tool. It was assigning the allLogs initiative at the intermediate root and running its remediation tasks, so the next review would have the record.

Watch for: workspaces created by individual teams "just for now", diagnostic settings that point to a storage account nobody reads, and alert rules whose action group emails a person who left. Resource Graph can find all three.

> *Visibility is the evidence clause of every other contract. Without it, governance is a set of claims nobody can check.*

CONTRACT CHECKLIST

- One central workspace in the management subscription; any extra workspace has a written reason.
- Activity logs from every subscription reach the central workspace, and table retention matches the required record-keeping period.
- A built-in diagnostic settings initiative is assigned at the intermediate root, with remediation tasks run.
- Every platform alert, including Service Health, has a severity, an action group and an owner.
- A workbook per environment shows compliance, posture, alerts, untagged resources and spend.

See it live: the governance map at hub.jcardena.com/azure/governance.html

CHAPTER TEN

Cost: Every Number Adds Up



The plain point: every euro or dollar the estate spends has an owner who expected it, a limit someone agreed to, and a record that adds up to the invoice. Cost is not a monthly surprise for finance to explain. It is a guarantee layer, governed by the same contracts as security.

In the series foundation, the contract has a budget clause: how much a door may spend, per run and per period, and what happens at the limit. On Azure the doors are subscriptions and resource groups, and the clause is written with tags, budgets, alerts and commitments. The slogan for this layer fits it exactly: every number adds up. A 3% error in a cost report is the kind of quiet failure that makes leaders stop trusting the whole platform.

Start with ownership: tags that mean something

Chapter Four gave each workload its own subscription per environment, which already answers most cost questions: the subscription is the bill line. Tags answer the rest. We require a short set, enforced by policy from Chapter Six and defined in the appendix tag dictionary: `owner`, `costCenter`, `environment`, `workload` and `dataClassification`. The vending process stamps them on the subscription and its resource groups. Cost Management can then group and filter spend by tag, and, on Enterprise Agreement and Microsoft Customer Agreement accounts, its tag inheritance setting can apply subscription and resource group tags to usage records, so a resource created without tags still lands on the right line.

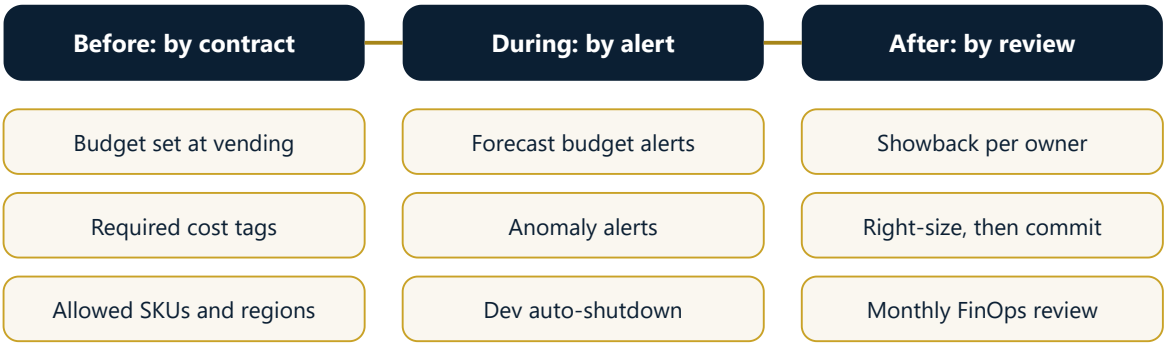
Showback comes first: each owner sees their own spend, every month, in the same view. Chargeback, where the cost moves to a budget line, can follow once the numbers are trusted. Trying chargeback on untrusted numbers starts arguments, not savings.

Shared platform costs need a rule too: the hub, the firewall and the central workspace otherwise become the line nobody owns. On Enterprise Agreement and Microsoft Customer Agreement accounts, Cost Management cost allocation rules can split the cost of a shared

subscription, resource group or tag across workload targets, by fixed percentages or in proportion to their own spend. Allocation changes what cost analysis, budgets and exports show, not the invoice, and it does not apply to purchases such as reservations and savings plans. For tools outside Azure, schedule Cost Management exports in the FOCUS format (FinOps Open Cost and Usage Specification), so finance can join Azure cost with other providers' data in one schema.

FIGURE 10.1

Cost controls before, during and after the spend



The cheapest cost incident is the one the contract refused before the resource existed.

Three moments, three kinds of control. Policy and vending stop the obvious mistakes; alerts catch the surprises within a day or two; the review turns patterns into commitments and new rules.

Budgets: an agreement with an early warning

A budget in Cost Management can be set at management group, subscription or resource group scope (and at billing scopes, depending on the agreement type). It resets each month, quarter or year, and sends notifications when actual cost or forecasted cost crosses a percentage threshold. A single budget can carry up to five thresholds.

Two facts shape how we use them. First, **a budget does not stop anything**; Microsoft is explicit that resources are not affected and consumption continues. Second, for Enterprise Agreement and Microsoft Customer Agreement accounts, cost data typically arrives within 8 to 24 hours (pay-as-you-go can take up to 72), and budgets are evaluated about once a day, so a budget is an early warning, not a circuit breaker. If you need automation at a threshold, a budget at subscription or resource group scope can call an Azure Monitor action group, for example to shut down dev virtual machines.

Our standard: every subscription gets a budget at vending time, with a forecast alert at 100% and actual alerts at 80% and 100%, sent to the owner tag and the platform cost channel. The budget is code, deployed with the subscription:

```
targetScope = 'subscription'

param amount int
param startDate string            // first day of a month, e.g. '2027-01-01'
param contactEmails array

resource budget 'Microsoft.Consumption/budgets@2023-05-01' = {
  name: 'budget-monthly'
  properties: {
    category: 'Cost'
    amount: amount
    timeGrain: 'Monthly'
    timePeriod: { startDate: startDate }
    notifications: {
      actual80: {
        enabled: true
        operator: 'GreaterThanOrEqualTo'
        threshold: 80
        thresholdType: 'Actual'
        contactEmails: contactEmails
      }
      actual100: {
        enabled: true
        operator: 'GreaterThanOrEqualTo'
        threshold: 100
        thresholdType: 'Actual'
        contactEmails: contactEmails
      }
      forecast100: {
        enabled: true
        operator: 'GreaterThan'
        threshold: 100
        thresholdType: 'Forecasted'
        contactEmails: contactEmails
      }
    }
  }
}
```

Anomaly alerts: the surprise detector

Budgets catch steady overspend. Anomalies catch the sudden spike: a scale setting left high, a test cluster that never stopped. Cost Management evaluates each subscription daily, comparing the day's usage with a forecast built from the previous 60 days, and runs about 36 hours after the end of the day so the data is complete. An anomaly alert rule sends an email once, at detection, with the resource groups that changed most.

Anomaly alert rules are created at subscription scope only, need Cost Management Contributor (or Contributor or Owner) on that subscription, and are limited to five per subscription. Alerts are sent using the current access of the rule's creator, so a rule created by a person whose role later expires under Privileged Identity Management (Chapter Five) goes silent. Microsoft's guidance for that case is to create the rule with a service principal through the Scheduled Actions API. We do exactly that, from the vending pipeline.

Commitments: for production, after right-sizing

Azure offers two ways to commit for one or three years in exchange for lower rates. A **reservation** commits to a resource type and configuration in a region; for virtual machines, instance size flexibility lets the discount apply to other sizes in the same group. It saves the most when the workload is stable. A **savings plan** commits to a fixed hourly spend across eligible services and regions: compute, plus a separate savings plan for select database services; it suits workloads that are still evolving.

Commitments live in the billing hierarchy, which is not the management group tree. An Enterprise Agreement has an enrollment, departments and accounts; a Microsoft Customer Agreement has a billing account, billing profiles and invoice sections. Both sit beside the tree and are governed by billing roles, not Azure RBAC. Map them on purpose: departments or invoice sections to cost owners, and the right to buy commitments to the FinOps lead. Each reservation or savings plan also has a scope: a resource group, a subscription, a management group, or shared across the billing profile or enrollment. Shared scope gives the best utilization; a narrower scope fits when one business unit funds the purchase.

Microsoft's own recommended order is the one we follow: right-size first, exchange or trade in underused reservations, buy new reservations only for stable, well-understood workloads, and add savings plans last, sized to a clean baseline. In contract terms: commitments are a production decision, approved in the FinOps review, never bought to cover waste in dev.

FIGURE 10.2

Cost rules by environment

Rule	Dev	Test	Prod
Budget with forecast alert	Yes, tight	Yes	Yes
Anomaly alert rule	Yes	Yes	Yes
VM auto-shutdown outside hours	Default on	On, with opt-out	Off
Allowed SKUs (policy)	Small list	Prod-like list	Approved list
Reservations and savings plans	No	No	After review
Diagnostic log category (Chapter Nine)	audit	audit	audit, plus allLogs added at vending

Cheap where it can be, deliberate where it must be. These are cost parameters, set per subscription at vending; the guardrails from Chapter Six stay identical in every environment. Logging depth is a cost decision too, which is why it appears on this page.

The FinOps rhythm

Tools find numbers; people decide. Once a month, the platform team, each workload owner and the FinOps lead look at the same view: spend against budget per subscription, anomalies and their causes, untagged spend, Azure Advisor cost recommendations, and commitment utilization. Each item ends with an owner and a date. Recurring causes become policy: if three teams left large dev clusters running, the allowed SKU list for dev gets shorter.

> **A COMPOSITE EXAMPLE: THE SILENT ALERT**
>
> An estate had anomaly alerts on every subscription, yet a large spike went unnoticed for a week. The rules had been created by an engineer whose elevated role was time-bound. When the role expired, the alerts stopped being sent, with no error anywhere. Recreating the rules under a pipeline identity, and adding "alert rule creator still has access" to the monthly review, closed the gap.

> *A budget is not a limit. It is a promise with an early warning. The limit is the contract you enforce before the money is spent.*

CONTRACT CHECKLIST

- Every subscription has owner, cost center, environment and workload tags, enforced by policy.
- Every subscription has a budget with actual and forecast alerts, deployed as code at vending.
- Anomaly alert rules exist on every subscription, created by a non-expiring pipeline identity.
- Reservations and savings plans are bought only for right-sized production, after review.
- A monthly FinOps review closes each item with an owner and a date.

See it live: the cost page at hub.jcardena.com/azure/governance-cost.html

PART V · OPERATE

CHAPTER ELEVEN

Deploy by Contract: Infrastructure as Code



The plain point: in production, nobody changes infrastructure by hand. Every change is code, reviewed by a person, previewed by the platform, deployed by a pipeline identity that has no password, and protected afterward so that the portal cannot quietly undo it. Here the model is the pull request: it proposes, and the contract decides.

Parts I to IV described what a governed estate looks like. Part V is about keeping it that way while dozens of teams change it every week. This chapter covers the door through which almost every change should arrive: the deployment pipeline. The next chapter covers the people and the rhythm that run the whole system.

Pick one language per layer, and use verified modules

Bicep and Terraform are both well supported on Azure. The contract does not care which you choose; it cares that you choose deliberately and do not mix them for the same resources. In a typical production estate, the platform layer (management groups, policy, hub network) and the workload layer may use different tools, but each resource has exactly one source of truth.

Do not write every resource from scratch. **Azure Verified Modules** (AVM) is a Microsoft initiative that sets one quality standard for infrastructure modules in Bicep and Terraform. Resource modules deploy a single resource with its common extensions, such as diagnostic settings, locks, role assignments and private endpoints; pattern modules deploy a reusable architecture. AVM states that it is owned by Azure engineering and supported by Microsoft. For Bicep, modules are published to the public registry and referenced by version:

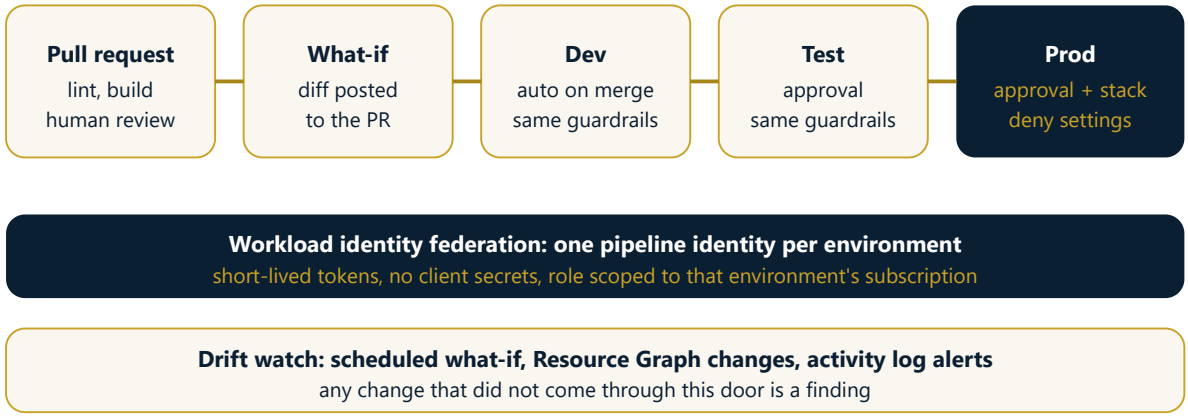
```
param kvName string

// AVM resource module, pinned to a published version (current at the time of writing).
module kv 'br/public:avm/res/key-vault/vault:0.14.2' = {
  name: 'kv-deploy'
  params: {
    name: kvName
    enableRbacAuthorization: true
    enablePurgeProtection: true
    publicNetworkAccess: 'Disabled'
  }
}
```

Pin versions and update them on purpose, through the same review as any other change. A module that floats to "latest" is a change nobody reviewed.

FIGURE 11.1

The deployment door, from pull request to production



Five gates, one identity per environment, one watch afterward. Humans review the intent; the platform previews the effect; approvals guard promotion; deny settings protect the result.

Pipelines without secrets

Chapter Five said no shared keys. Pipelines are where that rule is most often broken, because a service principal secret pasted into a pipeline variable works on day one and leaks on day four hundred. **Workload identity federation** removes the secret. The pipeline platform issues a short-lived token for a specific repository, branch or environment; Microsoft Entra ID trusts that issuer through a federated credential and exchanges the token for an Azure access token.

Azure DevOps now recommends workload identity federation for Azure Resource Manager service connections, with either an app registration or a user-assigned managed identity, and can convert existing secret-based connections. GitHub Actions uses the same mechanism through OpenID Connect with the Azure login action. Either way, our rules are the same:

- One identity per environment, so a dev pipeline cannot touch prod.
- The federated credential is bound to the protected branch or environment, not to any branch.
- Role assignments at the lowest scope that works, using the built-in roles from Chapter Five. Never grant "all pipelines" access to a prod connection.

What-if: the preview is part of the review

A reviewer reading Bicep sees intent. A reviewer reading the what-if output sees effect: which resources will be created, modified or deleted, and which properties change. Azure Resource Manager provides the what-if operation at resource group, subscription, management group and tenant scope (`az deployment group what-if` and its siblings), and the CLI can return JSON with `--no-pretty-print` for a pipeline to inspect.

We run what-if on every pull request against the target environment and post the summary into the review. Two cautions from Microsoft's documentation keep the gate honest: what-if can report "noise", properties that appear to change but will not, and it cannot evaluate some expressions such as `utcNow()` or references to resources outside the template. Treat the output as evidence for a person, and fail the gate automatically only on clear signals, such as an unexpected delete in prod.

Deployment stacks: protect what you deployed

A deployment stack is an Azure resource that manages a group of resources as one unit. It records what it manages, decides what happens to resources removed from the template (`detachAll`, `deleteResources` or `deleteAll`), and can apply **deny settings** that block changes from everyone else. The modes are `none`, `denyDelete` and `denyWriteAndDelete`. Stacks can be created at resource group, subscription or management group scope, and two built-in roles control them: Azure Deployment Stack Contributor and Azure Deployment Stack Owner.

```
az deployment group what-if \
  --resource-group rg-claims-prod \
  --template-file main.bicep \
  --parameters environment=prod

az stack group create \
  --name claims-prod \
  --resource-group rg-claims-prod \
  --template-file main.bicep \
  --parameters environment=prod \
  --action-on-unmanage detachAll \
  --deny-settings-mode denyWriteAndDelete \
  --deny-settings-excluded-principals "$EXCLUDED_GROUP_ID"
```

Two details matter. Deny settings apply only to the control plane, like the locks in Chapter Eight; they do not stop data plane operations such as writing a blob or a secret. And the excluded principals list accepts at most five entries, so follow Microsoft's recommendation and exclude a Microsoft Entra group that holds the pipeline identity, the break-glass administrators and the policy remediation identities. Forget the last one and your DeployIfNotExists policies from Chapters Seven and Nine will be refused on protected resources.

Policy is code too, and it is tested

Policy definitions, initiatives, assignments and exemptions live in the same repository and pipeline as everything else. Each change is versioned and reviewed, and rollback is a redeploy of the previous commit. Before a policy or role change reaches the production tree, test it. Microsoft's landing zone guidance describes a canary approach: a duplicate management group hierarchy with its own subscriptions and its own pipeline identities, where changes are deployed first against sample workloads. Microsoft notes it is optional and suits organizations with strong change management. Smaller estates can rely on the staging from Chapter Six instead: DoNotEnforce first, then a narrow resource selector, then the full scope. Open-source tooling such as Enterprise Policy as Code (EPAC) or the Azure landing zone modules can handle the deployment mechanics.

Drift: changes that did not come through the door

Even with every control above, something will change outside the pipeline: an emergency fix, a portal click by someone with elevated access. We look for drift three ways. A scheduled what-if against each production stack reports any difference from code. Azure Resource Graph keeps 14 days of resource configuration changes for investigation. An activity log

alert, from Chapter Nine, fires on write operations in prod whose caller is not a pipeline identity. Each finding is resolved one of two ways: the change is written back into code, or the code is redeployed over it.

No manual changes in prod does not mean no humans in prod. People hold Reader by default and elevate through Privileged Identity Management (Chapter Five) for incidents. An activation is an access event, recorded by PIM. Any manual change made during it is drift: it gets a change record and is written into code or reverted within an agreed time.

A COMPOSITE EXAMPLE: THE FIX THAT KEPT COMING BACK

An on-call engineer raised a database's capacity through the portal during an incident. The next routine deployment set it back, and the incident returned. The cause was not the engineer or the pipeline. It was a missing step: emergency changes were not written back into code. The team added the drift check and a rule that the incident is not closed until the pull request is merged.

> *Infrastructure as code is not about typing less. It is about making every change pass through a door that reviews, previews, records and protects it.*

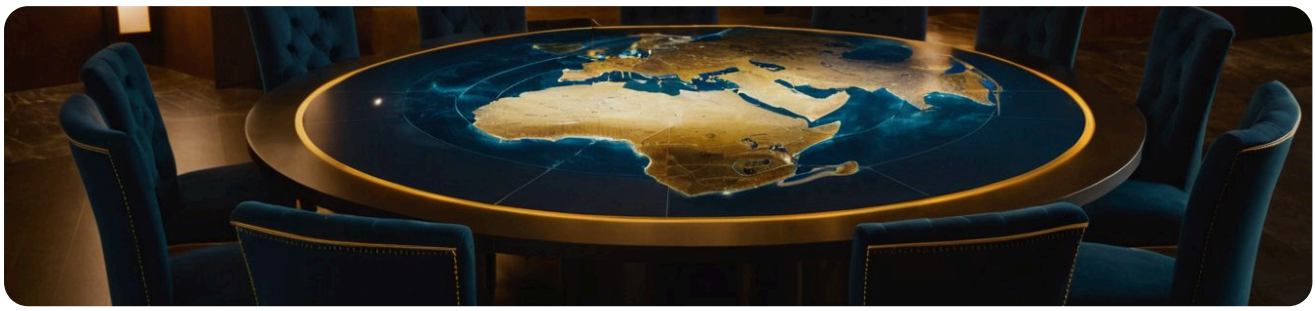
CONTRACT CHECKLIST

- Each resource has one source of truth in code; module versions are pinned.
- Pipelines use workload identity federation, one identity per environment, no secrets.
- Every pull request shows a what-if against its target environment.
- Promotion to test and prod needs an approval; prod resources sit in stacks with deny settings.
- Drift is detected on a schedule and every finding is closed in code.
- Policy changes are versioned, staged or tested in a canary hierarchy, and released through the pipeline.

See it live: the policy page at hub.jcardena.com/azure/governance-policy.html

CHAPTER TWELVE

The Governance Operating Model



The plain point: governance is not a set of settings you deploy once. It is a small team, a clear split of who decides what, a regular rhythm, and a one-page scorecard that tells leadership whether the estate is keeping its promises. Without those, every control in this book slowly decays.

Eleven chapters built the contracts: tenant, tree, subscriptions, roles, policy, network, posture, visibility, cost and deployment. This chapter answers the question every CIO asks at the end of a design review: who runs this, how often, and how will I know it is working? It also gives a 30-60-90 day plan for the case most readers actually have: not a clean new tenant, but an existing estate that grew without a plan.

Two kinds of team, one boundary

The Azure landing zone model separates a **platform team** from **workload teams**. The platform team owns everything under the Platform management group from Chapter Three (identity, management, connectivity), the policy set from Chapter Six, the subscription vending process from Chapter Four and the pipelines that deploy them. Workload teams own what lives inside their landing zone subscriptions: their spoke, their services, their data, their alerts and their spend.

The boundary between them is the contract. Workload teams move fast inside it without asking permission. When they need something outside it, a new region, a public endpoint, a SKU not on the list, they ask through one door: an exemption or a change to the contract, recorded with an owner, a reason and an expiry date.

FIGURE 12.1

RACI for the governed estate

Activity	Platform	Workload	Security	FinOps	Steering
Management groups and policy set	R/A	C	C	C	I
Subscription vending	R/A	C	I	C	I
Hub network, DNS, firewall	R/A	I	C	I	I
Resources in a landing zone	C	R/A	C	I	I
Posture findings in a landing zone	C	R	A	I	I
Budgets and commitments	R	C	I	A	I
Policy exemptions	R	C	A	C	I
Patching of workload machines	C	R/A	C	I	I
Changes to the contract itself	R	C	C	C	A

R responsible, **A** accountable, **C** consulted, **I** informed. One accountable owner per row. Steering is a small group chaired by the CIO or a delegate.

The rhythm

- **Daily, automated.** Policy evaluates, Defender assesses, anomaly detection runs, alerts reach their owners. Nobody meets.
- **Weekly, 30 minutes, platform team.** New exemption requests, drift findings from Chapter Eleven, failed remediation tasks, vending requests.
- **Monthly, one hour, platform plus workload owners, security and FinOps.** The scorecard, the FinOps review from Chapter Ten, top posture controls from Chapter Eight, exemptions that expire this month.
- **Quarterly, steering.** Contract changes: new policies moving from audit to deny, new allowed regions or services, commitment purchases, restore and failover test results.

The one-page scorecard

Leaders do not need fifty dashboards. They need about ten numbers, each tied to a guarantee from this book, each with a target and a trend. Every number below can be computed from Azure Policy compliance data, Defender for Cloud, Cost Management, Azure Resource Graph or the central workspace.

FIGURE 12.2

Scorecard: one row per guarantee

Layer	Measure	Source	Example target
Contracts	Policy compliance, prod scope	Azure Policy	Rising, set by steering
Contracts	Exemptions past expiry	Policy exemptions	Zero
Security	Secure score (model stated)	Defender for Cloud	Rising each month
Security	Standing privileged roles held by people	Entra ID, PIM	Emergency accounts only
Security	Prod workloads with tested restore this quarter	Review record	All
Visibility	Subscriptions sending activity logs centrally	Resource Graph	All
Visibility	Prod changes not made by a pipeline	Activity log	Each one explained
Cost	Subscriptions with budget and anomaly alert	Cost Management	All
Cost	Spend without owner tag	Cost analysis	Falling to near zero
Cost	Spend against budget, by owner	Cost Management	Within agreed range

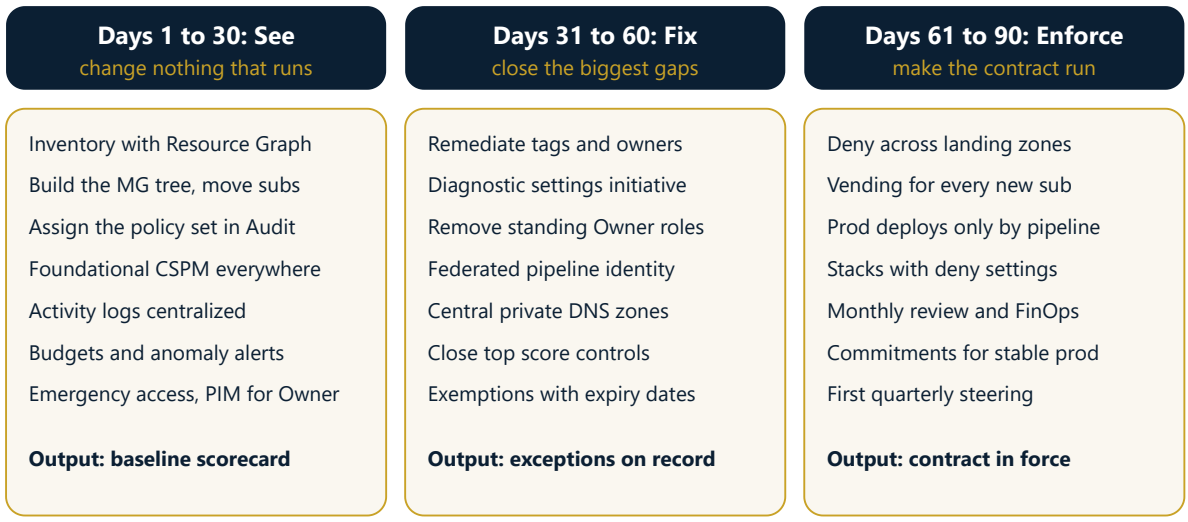
Targets are set by your steering group, not by this book. The point is that each number exists, is computed the same way every month, and has an owner.

30-60-90 days for an existing estate

Most estates already have dozens of subscriptions, owners nobody remembers and a few public endpoints that "must stay". The order that works is the same every time: see first, fix second, enforce third. Turning on deny policies on day one breaks deployments and spends the trust you need for the next ninety days.

FIGURE 12.3

See, fix, enforce



Audit before deny, always. The audit results from the first month tell you which deny rules will break something, so you can fix or exempt first and enforce without surprises.

Three practices make the plan survive contact with a real estate. First, publish the baseline scorecard at day 30 even if it looks bad; a bad honest number is the starting point leaders need. Second, give every exception an expiry date, not a promise; the weekly review chases the dates. Third, move rules from audit to deny one at a time, starting with the ones that carry the most risk and break the fewest deployments, typically public network access on data services and allowed regions.

Onboarding a new workload

1. The workload owner requests landing zones through the vending form: name, owner, cost center, data class, environments.
2. Vending creates one subscription per environment under the right management group, with tags, budget, anomaly alert, spoke network, diagnostic settings and Defender plans already set.
3. The platform team grants the team's groups their roles through PIM, and creates one federated pipeline identity per environment.
4. The team reads the contract summary: what is denied, what is audited, how to request an exemption.
5. The workload appears on the scorecard from its first day.

The contract evolves

Microsoft renames services, adds policy definitions and changes defaults; this book noted several of those changes in passing. The contract keeps up because it has an owner and a review, not because someone remembers. Each quarter, steering reviews new built-in policies, retired features and the exemption list, and each change ships through the same pipeline as everything else.

> *Trust, earned at every layer, is not a slogan you print. It is a scorecard you can show every month, with the same numbers computed the same way.*

CONTRACT CHECKLIST

- Platform and workload responsibilities are written in a RACI with one accountable owner per row.
- Weekly, monthly and quarterly reviews are on the calendar with named attendees.
- A one-page scorecard exists, with sources and owners for every number.
- Existing estates follow see, fix, enforce; no deny rule goes live before its audit results are reviewed.
- New workloads arrive only through vending, already inside the contract.

See it live: the governance map at hub.jcardena.com/azure/governance.html

APPENDIX

Appendix: Templates



These templates are the book on one desk. Copy them, change the names, argue about the cells with your team, and keep the result under version control next to the code it governs. A template that nobody disagrees with has not been read.

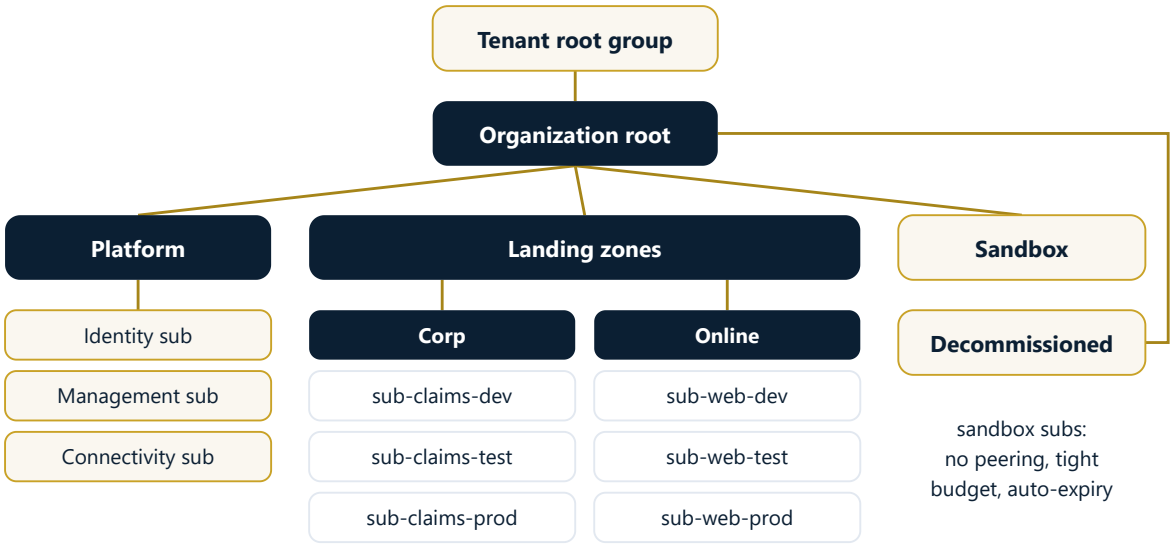
Each template points back to the chapter that explains it. Where a cell names a built-in Azure role or policy, check the current name in Microsoft's documentation before you rely on it; names and defaults change, and the pattern is what matters.

A.1 Management group and subscription map

The shape from Chapters Three and Four: a shallow tree, platform separated from workloads, one subscription per workload per environment. Replace the example workloads with your own.

FIGURE A.1

Management group and subscription map



Policy and roles inherit down the tree. Assign broad rules at Organization root and the guardrails at Landing zones, identical for every environment. Cost parameters and production-only obligations go on the workload subscriptions at vending. Keep the tree shallow; three or four levels below the root is plenty for most estates.

A.2 Role matrix: persona by environment

From Chapter Five: assign to groups, not people; the lowest scope that works; time-bound elevation through Privileged Identity Management (PIM) for anything that can change production. "Eligible" means the role is activated through PIM with a reason and an expiry.

Persona	Dev	Test	Prod
Workload developer	Contributor (eligible, no approval)	Reader; Contributor (eligible)	Reader; no write
Workload lead	Contributor (eligible)	Contributor (eligible)	Contributor (eligible, approval)
Pipeline identity (one per env)	Contributor on its sub	Contributor on its sub	Contributor on its sub
Application runtime	Managed identity with data roles only, for example Key Vault Secrets User		
Platform engineer	Owner eligible at the Landing zones management group; separate eligible roles at the Platform management group or its subscriptions, least privilege (approval required, at least two approvers configured)		

Access administrator	Role Based Access Control Administrator (eligible, with conditions limiting which roles it may assign)
Security team	Security Reader standing; Security Admin (eligible)
FinOps lead	Cost Management Reader standing; Cost Management Contributor eligible for budget changes; budgets and anomaly alert rules are created by the vending pipeline identity (Chapter Ten)
Auditor	Reader, time-bound access review
Emergency access accounts	Global Administrator in Microsoft Entra ID, used only in emergencies as Chapter Two describes; every sign-in raises an alert

Pipelines that also assign roles or deploy deployment stacks with deny settings need more than Contributor. Grant exactly that (for example a constrained Role Based Access Control Administrator assignment, or Azure Deployment Stack Owner) and record why.

A.3 Policy starter set with target effects

From Chapters Six to Ten. The guardrails are the same for dev, test and prod. Deny rules are assigned once at the Landing zones management group; Modify, DeployIfNotExists and audit baselines are assigned at the intermediate root, so platform subscriptions are covered too (Chapters Six and Nine). The only deliberately looser set is Sandbox. For an existing estate, every Deny row starts in Audit (or with enforcement set to DoNotEnforce) through the see and fix phases of the 30-60-90 plan (Chapter Twelve), then switches to the target effect on a date on the calendar. DINE means DeployIfNotExists.

Rule (intent)	Built-in starting point	Sandbox	Landing zones: dev, test and prod (target)
Resources only in approved regions	Allowed locations	Deny	Deny
Required tags present	Require a tag on resources	Audit	Deny
Tags inherited from subscription	Inherit a tag from the subscription if missing	Modify	Modify
Block risky resource types	Not allowed resource types	Audit	Deny
Approved VM sizes	Allowed virtual machine size SKUs	Audit	Deny, assigned per subscription at vending with an environment list
Every workload subnet has a network security group (special	Subnets should be associated with a	AuditIfNotExists	AuditIfNotExists

subnets such as GatewaySubnet and AzureFirewallSubnet excepted)	Network Security Group		
Public web apps behind a WAF	WAF built-ins for Front Door and Application Gateway	Not assigned	Audit, then Deny (Online)
Production VMs backed up	Azure Backup should be enabled for Virtual Machines	Not assigned	AuditIfNotExists; configure at prod subscriptions
No public IPs on VM NICs	Network interfaces should not have public IPs	Audit	Deny
Public network access off on data services	Per service, built-in where available	Audit	Deny
No `privatelink` DNS zones in spokes	Custom or landing zone reference	Not assigned	Deny
Private endpoint DNS records	Configure Azure PaaS services to use private DNS zones (landing zone initiative)	Not assigned	DINE
Resource logs to central workspace	Enable audit or allLogs category group resource logging to Log Analytics	DINE (audit)	DINE (audit at the intermediate root; allLogs added at production subscriptions during vending)
Defender for Cloud plans on	Defender for Cloud configuration policies	Foundational	Plans per Chapter Eight
Key Vault uses Azure RBAC and purge protection	Key Vault built-ins	Audit	Deny

Every exemption from this table carries an owner, a reason, a category (waiver or mitigated) and an expiry date. Expired exemptions appear on the scorecard.

A.4 Tag dictionary

From Chapters Four and Ten. Keep it short; every extra required tag is friction. Values come from controlled lists wherever possible.

Tag	Meaning	Example value	Required on	Enforced by

`owner`	Accountable group or role, not a person's name	claims-platform-team	Subscription, resource group	Policy deny + inheritance
`costCenter`	Finance code that pays	CC-4410	Subscription, resource group	Policy deny + inheritance
`environment`	dev, test, prod or sandbox	prod	Subscription, resource group	Vending + policy
`workload`	Short workload name	claims	Subscription, resource group	Vending
`dataClassification`	Highest sensitivity held: public, internal, confidential or restricted	confidential	Subscription, resource group, and resources holding data	Policy audit first, then deny
`rpoRto`	Recovery targets agreed with the owner	RPO 15m / RTO 4h	Prod subscription	Review (Chapter Eight)
`expiresOn`	Planned end date for temporary resources	2027-03-31	Sandbox, experiments	Policy audit + cleanup job

A.5 Readiness checklist

Use this before you declare a landing zone ready for its first production workload. Every item maps to a chapter; a "no" is a finding with an owner and a date, not a blocker by default.

READINESS: PLATFORM

- One tenant, emergency access accounts tested, MFA and conditional access in force (Chapter Two).
- Management group tree deployed from code; no subscriptions left under the tenant root (Chapter Three).
- Vending creates subscriptions with tags, budget, anomaly alert, spoke and logging (Chapters Four, Nine, Ten).
- No standing Azure Owner for people; emergency accounts hold permanent Entra Global Administrator only; PIM for privileged roles; access reviews scheduled (Chapter Five).
- Policy starter set assigned, audit results reviewed, deny live across landing zones (Chapter Six).

READINESS: GUARANTEE LAYERS

- Hub, firewall, resolver and central `privatelink` zones in place; egress logged (Chapter Seven).
- Defender for Cloud on; score model chosen; required compliance standards assigned; locks on critical platform resources (Chapter Eight).
- RPO, RTO, recovery region and a failover test recorded for each production tier (Chapter Eight).
- Central workspace receiving activity and resource logs; platform alerts routed to owners (Chapter Nine).
- Budgets, anomaly alerts and the monthly FinOps review running (Chapter Ten).
- Pipelines on workload identity federation, what-if in review, stacks protecting prod (Chapter Eleven).
- RACI, rhythm and scorecard agreed by steering (Chapter Twelve).

> *Templates are where a philosophy becomes a habit. Fill them in, check them in, and review them on the calendar.*

CONTRACT CHECKLIST

- Each template above is copied into your repository and owned by a named team.
- Every built-in role and policy name in your copy has been checked against current Microsoft documentation.
- The policy table shows the target effect for Sandbox and for landing zones, and the audit-to-deny dates are on the calendar.
- The readiness checklist is completed before the first production workload goes live.

See it live: the landing zone map at hub.jcardena.com/azure/governance-landing-zone.html

THE SERIES

GDCD on Azure

GDCD on Azure applies Governance-Driven Contract Design to one platform, end to end. The first book governs the estate. The second delivers AI on top of it.

Book 1	**Governing Azure by Contract**: a layer-by-layer guide to a governed Azure platform. (This book.)
Book 2	**AI Foundry by Contract**: delivering governed AI on Azure.

The foundation of the method is *The Contract Decides: Governing AI at Every Door*, Book 1 of the GDCD Field Guides. See every Azure service, model and region in the live diagrams at hub.jcardena.com/azure. All books are free at blog.jcardena.com/en/ebooks.

Juan Cardena

Enterprise AI & Agentic Systems Architect · blog.jcardena.com

Download the latest edition at blog.jcardena.com/en/ebooks/governing-azure-by-contract. See it live at hub.jcardena.com/azure.

GDCD on Azure · Book 1 · Edition 1.1 · Trust, earned at every layer.