

GDCD ON AZURE

BOOK 2

GOVERNANCE-DRIVEN CONTRACT DESIGN

AI Foundry by Contract

Delivering governed AI on Azure.

AI on Azure is easy to start and hard to govern. A model deployment takes minutes; knowing who may call it, what it may see, what it costs and whether its answers are right takes design. This guide builds Microsoft Foundry on top of a governed landing zone, so every project, model, key, token and agent has an owner, a limit and a log. The model proposes. The contract decides.

Juan Cardena

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

CONTENTS

00 Why This Book

PART I · FOUNDATION

01 The Model Proposes, the Contract Decides

02 Where Foundry Lives in the Landing Zone

03 Roles for AI Work

PART II · BUILD

04 Models, Deployments and Quota

05 Private by Default for AI

06 Safety and Responsible AI

07 Every Answer Is Earned: Evaluations

PART III · RUN

08 Tracing and Visibility

09 Tokens Are Spend: The AI Gateway

10 Agents by Contract

A Appendix: Templates

PREFACE

Why This Book



Every enterprise estate now gets the same request, usually within a quarter of the platform going live: "We need AI, and we need it soon." The first pilot is easy. A team opens a portal, picks a model, copies a key into a configuration file and has a working demo by Friday. The hard part comes later, when there are twenty projects, three regions, a shared quota nobody planned, keys nobody can trace and a bill nobody can explain.

This book is about delivering AI on Azure without building that second picture. It is the second book in the GDCD on Azure series. The first, *Governing Azure by Contract*, built the platform: the tenant, the management group tree, one subscription per workload per environment, roles with an expiry, policy that runs, private networking, security posture, one place to see everything and cost that adds up. This book assumes all of that is in place, and puts Microsoft Foundry inside it.

The promise is short. When you finish, your platform team can stand up a Foundry setup where **every project, model, key and token has an owner, a limit and a log**. Nothing in that sentence depends on a model version or a portal layout. It depends on contracts.

Why AI needs the platform first

AI does not create new governance problems as much as it multiplies the old ones. A public endpoint on a storage account is a risk. A public endpoint on a model that can read that storage account, summarize it and send the summary somewhere is the same risk with a faster exit. A shared key in a pipeline is debt. A shared key that can spend tokens at machine speed is debt with interest. The platform controls from Book 1 are the floor this book stands on. Where a chapter needs one, it points back to it as "Governing Azure by Contract, chapter N" instead of repeating it.

FIGURE 0.1

Book 2 stands on Book 1



The AI layer inherits the platform. If a platform control is missing, no amount of care in the AI layer will make up for it. Fix the platform first.

The spine of the series

Every book in the series follows the same spine, which comes from the series foundation, *The Contract Decides*. Start from the business outcome. Turn it into guarantees. Carry each guarantee through four layers: quality, security, visibility and cost. Write it as a contract at every door into the system, and make the system enforce it, prove it and evolve it.

> **The model proposes. The contract decides.** *A model can suggest an answer, a tool call or a deployment. Whether that suggestion crosses a boundary is decided by identity, policy, network rules, gateway token limits and evaluations that the model cannot change.*

In practice that means a short list of questions for every door in an AI workload. Who may cross? What may cross? What must be true afterward? Who finds out? Book 2 answers those questions for the doors that Foundry adds to your estate: the Foundry resource, its projects, model deployments, quota, connections to your data, the gateway in front of the models and the agents that call tools on your behalf.

A note on names

Microsoft renames its AI services often. The product this book covers started as Azure AI Studio, became Azure AI Foundry and is now **Microsoft Foundry**. Azure AI Services is now called Foundry Tools. At the time of writing, the built-in roles have also been renamed: the roles once called Azure AI User, Azure AI Project Manager, Azure AI Account Owner and Azure AI Owner are now Foundry User, Foundry Project Manager, Foundry Account Owner and Foundry Owner, with the same role IDs. You will still see older names in portals, policy definitions and scripts for a while. Product names, built-in roles and feature status in this book were checked against Microsoft documentation in November 2026.

That is why each chapter teaches the pattern first and the product second. "Builders get data access to one project, not control of the whole resource" will still be true after the next rename. The role name that implements it may not. Where a feature is in preview, or a detail is likely to change, the text says "at the time of writing", and you should check the current Microsoft Learn page before you build on it. The book uses no prices of its own. Token, quota and threshold values in the samples are illustrative, and where the text cites a limit, the limit comes from Microsoft documentation.

Who this book is for

- **CIOs and CISOs** who need to say yes to AI without saying yes to an unknown risk. Read the opening paragraph and the checklist of each chapter.
- **Cloud architects and platform team leads** who will design the Foundry footprint. Read everything, including the snippets.
- **FinOps leads** who will be asked why the AI line on the bill doubled. Chapters 4 and 9 are yours.
- **AI engineering leads** who want their teams to move fast inside guardrails instead of waiting for approvals. Chapters 3, 7 and 10 show where the guardrails sit.

How the book is built

Part I, Foundation, places Foundry in the landing zone and decides who may do what. Part II, Build, covers models and quota, private networking, safety and evaluations. Part III, Run, covers tracing, the AI gateway and agents, ending with a path from pilot to production. The appendix collects the templates: a readiness checklist, a role matrix, an approved-model register, a gateway policy sample and an evaluation gate checklist.

Each chapter opens with its point in plain words, then gives the engineering detail. Each ends with a contract checklist you can copy into your own backlog, and a link to the live diagrams on the hub. None of this describes a product demo. It describes how a platform

team runs AI in production estates, written down so that your team can do the same.

CONTRACT CHECKLIST

- The Book 1 platform controls (identity, management groups, subscriptions, roles, policy, network, visibility, cost) are in place before the first Foundry resource.
- Every AI workload has a named business outcome and an owner.
- The team agrees to name things by pattern first and product second.
- Someone owns checking Microsoft Learn for renames and preview changes each quarter.

See it live: hub.jcardena.com/azure

PART I · FOUNDATION

CHAPTER ONE

The Model Proposes, the Contract Decides



A model is not a control. It can be accurate most of the time, careful most of the time and cheap most of the time, and none of that is a guarantee. Guarantees come from the system around the model: who may reach it, what data it may see, where it runs, how much it may spend and how its answers are checked. This chapter shows how to turn an AI business outcome into those guarantees before anyone deploys a model.

Most AI programs start the other way round. A team picks a model, builds a demo, and then asks the platform team to "make it production ready". By then the important decisions are made. The model was deployed with a key. The data source has a public endpoint. The deployment type sends prompts to any region. The quota is shared with every other team in the subscription. Making it production ready now means undoing work, and teams resist undoing work that already looks finished.

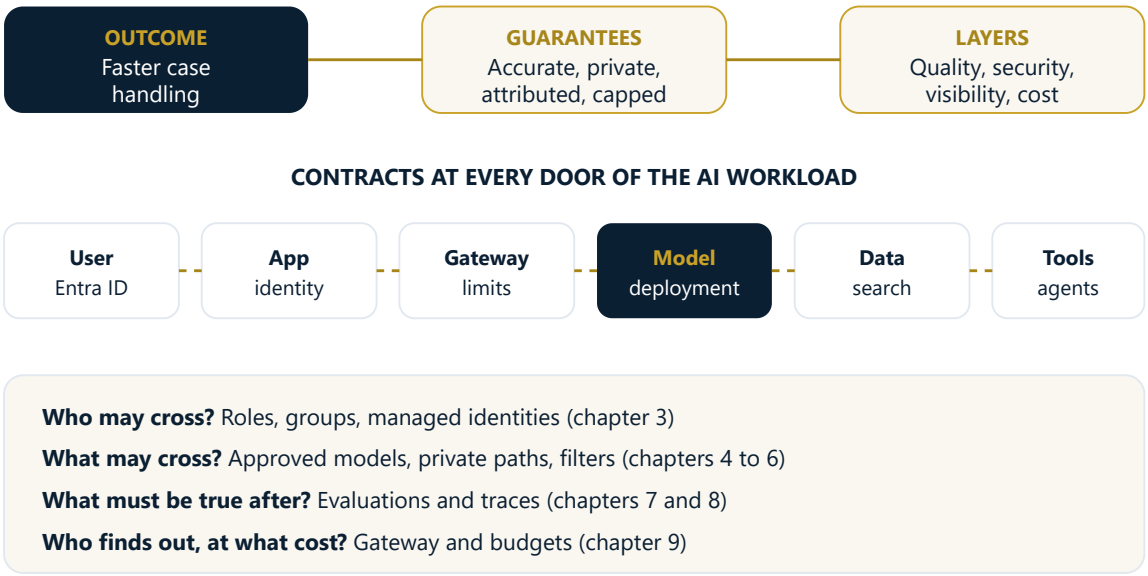
Start from the outcome

The series foundation, *The Contract Decides*, asks one question first: what does the business pay for? Not "a chatbot" or "an agent", but a result. Take a common enterprise pattern: a case-handling team wants an assistant that drafts a summary of each new case from the documents attached to it, so a person can review and decide faster. The outcome is shorter handling time with no drop in decision quality.

That outcome already implies guarantees. If decision quality must not drop, the summaries must be checked against the source documents. If the documents contain personal data, the data must stay in an approved geography and only reach approved people. If handling time is the goal, the assistant must answer within a set time. If the business case depends on cost per case, the token spend must have a ceiling. None of these are model features. All of them are contracts.

FIGURE 1.1

From outcome to contracts at every door



The model is one door of six. Every dashed line is a boundary with an owner. The model deployment is the door teams talk about most and govern least.

Five guarantees every AI workload needs

The same five guarantees appear in almost every enterprise AI workload. The wording changes. The shape does not.

FIGURE 1.2

Guarantee, layer, enforcer

Guarantee	Layer	Enforced by	Where
Only named identities reach the model	Security	Entra ID roles, key access disabled	Chapter 3; Governing Azure by Contract, chapter 5
Only approved models, in approved places	Security, cost	Azure Policy on deployments and deployment types	Chapter 4; Governing Azure by Contract, chapter 6
No public path to the model or its data	Security	Private endpoints, network policy	Chapter 5; Governing Azure by Contract, chapter 7
Answers meet a measured bar before release	Quality	Evaluation gates in the pipeline	Chapters 6 and 7
Every call is attributed and has a ceiling	Visibility, cost	Tracing, gateway token limits, budgets	Chapters 8 and 9; Governing Azure by Contract, chapter 10

Every guarantee names its enforcer. A guarantee with no enforcer is a wish. If a row in your version has an empty "enforced by" cell, that is the first backlog item.

Notice that three of the five rows point back to Book 1. That is deliberate. Foundry has its own controls, but it sits on Azure Resource Manager like every other service, so the platform contracts you already have apply to it. Azure Policy can deny a deployment. Role assignments decide who can create one. Diagnostic settings send its logs to the same workspace as everything else. You do not need a parallel governance program for AI. You need to extend the one you have.

Where the guarantees meet regulation

Buyers in regulated industries ask how these guarantees map to the frameworks their auditors use. The table below is orientation, not legal advice. Which obligations apply depends on the use case, the jurisdiction and the role your organization plays, and your legal and compliance teams make that call. What the table shows is that the evidence those frameworks ask for is the same evidence the contracts in this book already produce.

Framework	What it asks for, in short	Where this book produces the evidence
EU AI Act (Regulation (EU) 2024/1689)	Risk-based obligations. For high-risk systems: risk management, data governance, technical documentation,	Safety tiers and human approval (chapters 6 and 10), evaluations (chapter 7), traces as

	record-keeping, transparency, human oversight, and accuracy and robustness	records (chapter 8), the register and data rules (chapters 4 and 5)
NIST AI RMF 1.0	A voluntary framework organized in four functions: Govern, Map, Measure and Manage	Govern: contracts, roles and the register (chapters 1, 3 and 4). Map: outcome and tier at intake (chapters 1 and 6). Measure: evaluations and tracing (chapters 7 and 8). Manage: gateway, agents and quarterly review (chapters 9 and 10)
ISO/IEC 42001:2023	A certifiable AI management system: policy, roles, risk and impact assessment, lifecycle controls, monitoring and continual improvement	The templates in the appendix, with each chapter's contract checklist as the control record

Why the platform must come first

AI multiplies whatever governance you already have. In an estate with clean subscriptions per environment, a Foundry resource in the production subscription inherits production policy, production budgets and production access reviews on day one. In an estate where dev and prod share a subscription, the same resource shares quota, keys and blast radius with every experiment in the building.

> **The fastest path is not the shortest one.** A pilot that skips the landing zone looks faster for a month. Then it has to be rebuilt with private networking, managed identity and a new subscription, and the outbound network choices for agents cannot be changed after the Foundry resource is created (chapter 5). Building it inside the guardrails from the start is cheaper than moving it later.

Writing the first AI contract

Before the first deployment, write a one-page contract for the workload. It does not need a special format. It needs to be short enough for a person to read in a minute and specific enough for a machine to check most of it.

```
contract: case-summary-assistant
owner: case operations lead
environment: prod
outcome: shorter handling time, no drop in decision quality
models: from approved register only (chapter 4)
data_processing: data zone matching the data residency rule
access:
  callers: app managed identity only; no keys
  builders: case-ai-builders group, project scope only
network: no public endpoint on model, search or storage
quality: groundedness on the release test set meets the agreed bar (chapter 7)
spend: monthly token ceiling at the gateway; alert at 80%
evidence: every call traced with caller, model, tokens, latency
```

Every line in that contract maps to a control in a later chapter. When you read the rest of this book, read it with your own contract open, and fill in the enforcer for each line.

CONTRACT CHECKLIST

- The AI workload has a written business outcome and a named owner.
- Each guarantee has a layer and a named enforcer, not a person's promise.
- The workload will live in a subscription per environment from Book 1, never a shared dev-and-prod one.
- A one-page contract exists before the first model deployment.
- Empty "enforced by" cells are on the backlog with an owner and a date.

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

CHAPTER TWO

Where Foundry Lives in the Landing Zone



Foundry is not a special place outside your estate. It is an Azure resource, and it belongs in the same management group tree, the same subscriptions per environment and the same policy scope as everything else you run. Put it in the right place on day one and most of its governance comes for free. Put it in the wrong place and every later control is a workaround.

The resource model, in plain words

At the time of writing, Microsoft Foundry has two ways to organize work. The current one is a **Foundry resource** with **projects** inside it. The older one is a hub with hub-based projects, built on Azure Machine Learning, and it is now shown in the Foundry (classic) portal. Microsoft states that new investment is focused on Foundry projects in the new portal. For new work, we use the Foundry resource model and treat hubs as something to migrate away from, with one caution: at the time of writing, some capabilities, such as deploying certain open models to managed compute, still point to hub-based projects. Check the current docs for the features your workload needs.

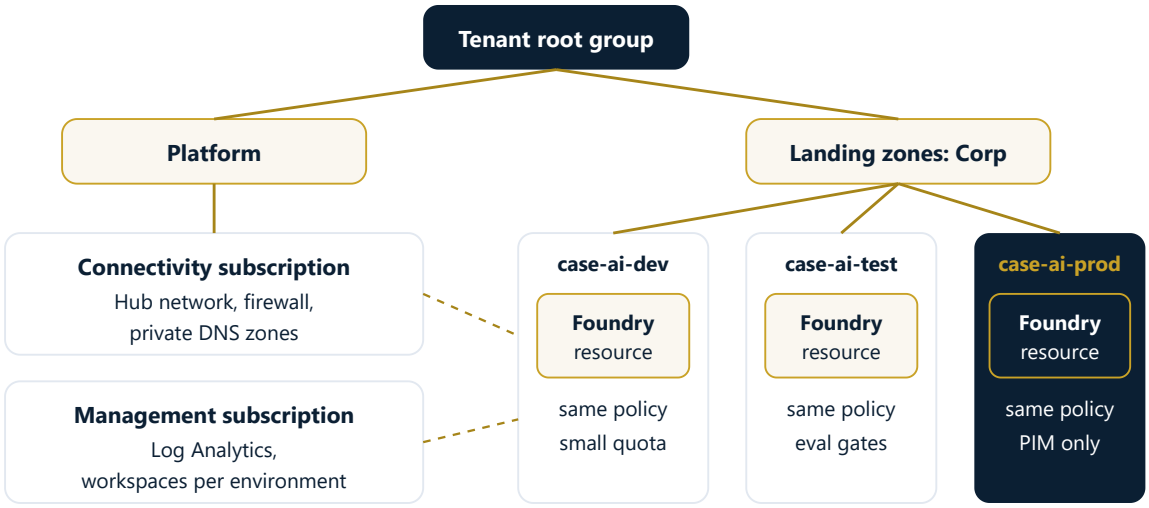
Under the hood, a Foundry resource is an Azure AI Services account (`Microsoft.CognitiveServices/accounts`, kind `AIServices`) with project management turned on. Three things hang from it:

- **The resource itself** is the administrative, security and monitoring boundary. Networking, key settings, encryption and diagnostic settings live here.
- **Projects** (`accounts/projects`) are child resources where teams build. Each project has its own managed identity, its own role assignments and its own endpoint.
- **Model deployments** (`accounts/deployments`) are also children of the resource, not of a project. Every project in the resource can use them.

That last point shapes the whole design. Because deployments and network settings are shared across projects, **the Foundry resource is the real isolation boundary**. A project is a workspace and an access scope. It is not a wall.

FIGURE 2.1

Foundry in the management group tree



One Foundry resource per workload per environment

Projects inside each resource separate teams or use cases, not environments.
Dashed lines: private DNS from Connectivity, logs to Management.

Foundry follows the tree you already have. Policy on the Corp management group reaches every Foundry resource beneath it, with the same guardrails in dev, test and prod. What differs is quota, access and promotion gates, not the rules. Production never shares a resource, a quota pool or a key with dev.

Two placement patterns

In practice we see two patterns, and both can be governed well. What matters is that the choice is written down.

Pattern	How it works	Strength	Watch for
Resource per workload per environment	Each workload gets a Foundry resource in its own dev, test and prod subscriptions	Clear cost, clear blast radius, quota follows the subscription	More resources to keep consistent; needs infrastructure as code
Shared AI platform per environment	The platform team runs one Foundry resource per environment; workload teams get projects	Central model approval, fewer deployments to manage	Shared network settings and deployments; cost needs a gateway or tags to split

Our default is the first pattern for any workload that handles sensitive data or has its own budget line, and the second for internal productivity use cases that share the same data rules. In both, the environment split from Governing Azure by Contract, chapter 4, is not optional: dev, test and prod live in different subscriptions. Quota is assigned per subscription (chapter 4), so this split also stops a dev experiment from consuming production capacity.

Region and data residency

Region choice for AI has two parts, and teams often confuse them. The first is where the Foundry resource lives, which decides where the data that the Foundry resource itself stores is kept at rest. Connected services such as storage, search, Cosmos DB and Application Insights keep their data in their own regions, so each one needs its own location decision. Microsoft documents that data stored at rest remains in the designated Azure geography. The second is where prompts are processed, which is decided by the **deployment type** of each model deployment, not by the resource location. A Global deployment may process prompts in any Azure region. A Data Zone deployment keeps processing inside the Microsoft-defined data zone, such as the US or the EU. A Standard or Regional Provisioned deployment keeps it inside the Azure geography you choose, although Microsoft notes it may move between regions within that geography for operational purposes. Chapter 4 turns this into policy.

So the region contract has two clauses: allowed regions for the resource, enforced by the allowed-locations policy you already run (Governing Azure by Contract, chapter 6), and allowed deployment types per environment, enforced by a deployment policy. Also check that the models you need are available in your chosen region and deployment type before you commit. Availability differs by model and changes often.

Deploy it as code

Create Foundry resources through the same pipelines as the rest of the estate (Governing Azure by Contract, chapter 11), never by hand in production. A minimal production shape in Bicep looks like this. The API version is the one used in Microsoft's published samples at the time of writing.

```
resource foundry 'Microsoft.CognitiveServices/accounts@2025-06-01' = {
  name: foundryName
  location: location
  kind: 'AIServices'
  sku: { name: 'S0' }
  identity: { type: 'SystemAssigned' }
  properties: {
    allowProjectManagement: true
    customSubDomainName: foundryName
    disableLocalAuth: true
    publicNetworkAccess: 'Disabled'
  }
}

resource project 'Microsoft.CognitiveServices/accounts/projects@2025-06-01' = {
  parent: foundry
  name: projectName
  location: location
  identity: { type: 'SystemAssigned' }
  properties: {}
}
```

Two lines carry most of the governance: `disableLocalAuth: true` removes key access, so every call must use Entra ID (chapter 3), and `publicNetworkAccess: 'Disabled'` means the resource is reachable only through private endpoints (chapter 5). Policy should enforce both, so the template is the convenient path and policy is the guarantee.

> **THE ORPHANED HUB**
>
> A common pattern in an existing estate: a hub created during an early pilot, in a subscription shared by several teams, with its own storage account, key vault and keys still active. It has no owner, and two production apps depend on it. The remedy is not a rebuild in place. It is a new Foundry resource in the right subscription, a migration of each app with an owner and a date, and then decommissioning the hub through the Decommissioned management group.

CONTRACT CHECKLIST

- Every Foundry resource sits in a workload or platform subscription under the right management group, one environment per subscription.
- The placement pattern (per workload or shared platform) is written down with its reason.
- New work uses Foundry resources and projects; any hub has an owner and a migration date.
- Allowed resource regions and allowed deployment types per environment are both written as policy.
- Production Foundry resources are created only through the pipeline, with key access and public access disabled.

See it live: hub.jcardena.com/azure/azure-ai-works.html

CHAPTER THREE

Roles for AI Work



AI work has three verbs: deploy, build and call. Deploying decides which models exist and how much capacity they get. Building means creating agents, prompts and evaluations inside a project. Calling means sending a request to a model or an agent and getting an answer back. Most access problems in AI come from giving one person, or one app, all three verbs when it needed one. This chapter assigns each verb to the smallest role that carries it.

The rules from Governing Azure by Contract, chapter 5, still apply without change: built-in roles first, assign to groups and not people, scope as low as possible, privileged roles through Privileged Identity Management with an expiry, regular access reviews and workload identities instead of secrets. What changes with Foundry is the role catalog and one important trap: keys.

Keys bypass every role

A Foundry resource can accept two kinds of authentication: Microsoft Entra ID tokens and API keys. Microsoft's documentation is direct about the difference: role-based access control applies when you authenticate with Entra ID, and a key grants full access without role restrictions. That access covers the data-plane APIs that accept key authentication. A key does not grant Azure Resource Manager rights such as role assignments, but where it works it ignores every role you designed. A key also carries no identity, so the log cannot say who called. Every guarantee in chapter 1 that starts with "only named identities" fails the moment a key is in use.

So the first contract of this chapter is simple: **no keys**. Set `disableLocalAuth` to true on every Foundry resource, and enforce it with the built-in policy listed at the time of writing as *Azure AI Services resources should have key access disabled (disable local authentication)*, which supports Audit and Deny. A companion DeployIfNotExists policy, *Configure Azure AI Services resources to disable local key access (disable local authentication)*, can remediate

existing resources. Policy display names in this area are being renamed toward "Foundry Tools", so search by the description if the name has changed. Assign it in Audit in every environment for a sprint to find what breaks, then switch to Deny everywhere: the same audit-first rule that Governing Azure by Contract, chapter 6, applies to every policy.

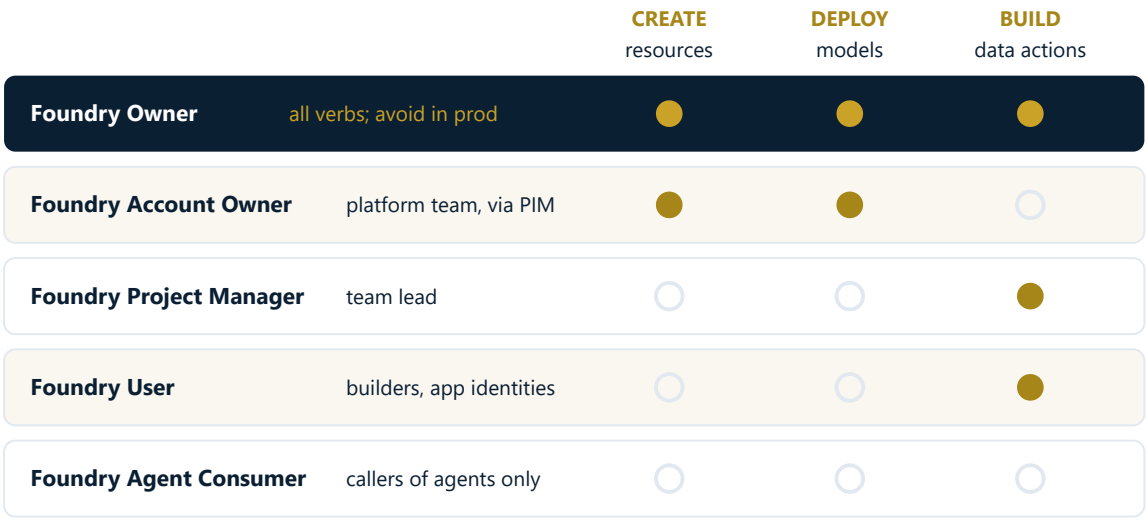
> **Tools that still expect keys.** Some older tools and samples assume key access and stop working when it is disabled. Find them in dev, move them to Entra ID or managed identity, and do not grant a production exemption to keep an old sample alive.

The Foundry role catalog

At the time of writing, Microsoft documents five Foundry built-in roles. They were recently renamed from "Azure AI" to "Foundry" names with the same role IDs, so use the role definition GUIDs in code, not the names.

FIGURE 3.1

Who gets which verb



Filled dot means the role carries the verb. The Account Owner can create and deploy but cannot build, so the people who control the platform do not quietly become builders. Agent Consumer can only call agent endpoints. Publishing agents is a separate permission: Foundry Project Manager and Foundry Owner carry it, Foundry Account Owner does not. Source: Microsoft Learn, role-based access control for Microsoft Foundry, at the time of writing.

Two more roles complete the set. **Cognitive Services Usages Reader**, assigned at subscription scope, lets people see quota without broader read rights; the Usages API ignores assignments at lower scopes. And the general Azure roles still matter: Microsoft documents that users with **Contributor** can deploy models in Foundry, and Owner can do

the same plus assign any role. A subscription Contributor is therefore a model deployer, whether you meant that or not. That is one more reason to keep standing Contributor off production subscriptions (Governing Azure by Contract, chapter 5).

Some role names look right and are not. Microsoft advises against the **Cognitive Services** roles for Foundry scenarios, except Usages Reader, and notes that **Azure AI Developer** applies to Azure Machine Learning workspaces and Foundry hubs, not to Foundry projects. **Cognitive Services OpenAI User** and **OpenAI Contributor** still exist, but they belong to standalone Azure OpenAI resources. If you still run some, keep them on those roles until you upgrade them to Foundry resources, then move to the Foundry roles.

A role matrix per persona and environment

This is the single role matrix for the book; Appendix B repeats it unchanged. Note the deployment pipeline: in Microsoft's permission table, only Foundry Account Owner (or Foundry Owner) can manage model deployments, and only Foundry Project Manager (or Foundry Owner) can publish agents, so the pipeline identity holds both narrower roles instead of Foundry Owner.

Persona	Dev	Test	Prod
Platform team	Foundry Account Owner, standing	Foundry Account Owner via PIM	Foundry Account Owner via PIM, approval required
Use case lead	Foundry Project Manager	Foundry Project Manager via PIM	Reader; Foundry User via PIM for incidents
Builders	Foundry User on the project; Reader on the resource	Reader; Foundry User on the project via PIM for debugging	Reader only
Deployment pipeline (workload identity)	Foundry Account Owner to deploy models; Foundry Project Manager to publish agents	Same	Same, from the protected release pipeline only
Gateway or model-calling app	Foundry User on the Foundry resource, or a custom inference-only role	Same	Gateway identity only; apps call the gateway (chapter 9)
Agent-calling app	Foundry Agent Consumer at project or agent scope	Same	Same, agent scope preferred
Project identity	Foundry User on the Foundry resource (needed for continuous evaluation)	Same	Same

Responsible AI owner, auditor	Reader; Log Analytics Reader	Same	Same; content tables through PIM
FinOps lead	Cognitive Services Usages Reader at subscription scope, plus Cost Management Reader as in Governing Azure by Contract, chapter 10		

Two details matter here. First, Microsoft documents that calling a model with Entra ID at inference time needs Foundry User on the Foundry resource, which also carries build data actions. If that is broader than the contract allows, build a custom role with only the data actions the workload needs. Microsoft documents how. Second, Foundry Agent Consumer covers agent endpoints only; it does not allow direct model inference. It can be assigned at the scope of a single agent. At the time of writing, the Azure portal assigns it only at resource scope, so use the CLI or infrastructure as code for project or agent scope.

Apps use managed identities

Every app that calls a model runs as a managed identity, never as a user and never with a key. Each Foundry project also has its own managed identity, which it uses to reach connected resources such as storage and search; give that identity only the data roles it needs on those resources, such as Storage Blob Data Reader. Assign roles in code, by GUID:

```
# Foundry User (role ID 53ca6127-db72-4b80-b1b0-d745d6d5456d)
# to the builders group, at project scope
az role assignment create \
  --assignee-object-id "$BUILDERS_GROUP_ID" \
  --assignee-principal-type Group \
  --role "53ca6127-db72-4b80-b1b0-d745d6d5456d" \
  --scope "$FOUNDRY_ID/projects/$PROJECT_NAME"
```

> **THE FINE-TUNING REQUEST**
>
> A team asks for Foundry Owner in production "to fine-tune a model". Fine-tuning needs both data plane and control plane rights, and Foundry Owner is the only built-in role with both. The contract answer is to split it: builders fine-tune in dev with Foundry User and Account Owner through PIM, and the resulting model reaches production through the pipeline, not through a person, after it enters the approved-model register and passes the evaluation gate like any other model (chapters 4 and 7).

CONTRACT CHECKLIST

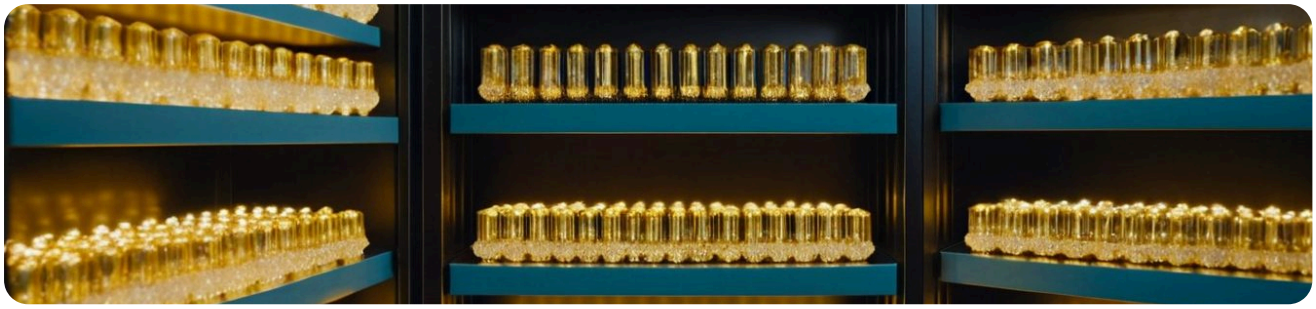
- Key access is disabled on every Foundry resource and denied by policy.
- Roles are assigned to groups or managed identities, by role ID, through code.
- People hold Foundry Account Owner and any production access only as PIM-eligible roles; the pipeline identity carries the rights to deploy models and publish agents.
- No standing Owner or Contributor on production subscriptions, since both can deploy models.
- Callers that only use agents have Foundry Agent Consumer, scoped to the project or the agent.
- Access reviews cover Foundry roles on the same cycle as the rest of the estate.

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

PART II · BUILD

CHAPTER FOUR

Models, Deployments and Quota



A model in the catalog is an offer. A model deployment is a commitment: it has a location for processing, a billing model, a share of capacity and an owner who answers for it. Treat every deployment as a contract with four clauses. Which model, approved by whom. Where prompts are processed. How much capacity it may use. When it retires.

The catalog is large. At the time of writing, Microsoft describes more than ten thousand models in it, with new ones every month. No enterprise should let all of them reach production. The aim of this chapter is not to pick the best model. It is to make sure that whichever model a team picks went through the same door.

Two kinds of model in one catalog

Microsoft organizes the Foundry model catalog into two groups, and the difference matters for governance.

	Models sold by Azure	Models from partners and community
Terms	Hosted and sold by Microsoft under the Microsoft Product Terms	The provider's license terms; Non-Microsoft Products
Billing	Azure meters	Azure Marketplace
Support	Microsoft	The provider
Review	Microsoft Responsible AI review, summarized on the model card	Typically validated by the provider

Neither group is automatically safe or unsafe. Microsoft itself says its reviews do not guarantee that risks are mitigated, and that customers should run their own evaluations. What the table changes is the paperwork: legal review of license terms, procurement for

Marketplace billing, and a support path for incidents. A model from the partner group needs those three settled before it reaches the register.

The approved-model register

The register is a short, owned list of the models each environment may deploy. One row per model and version: the source group, the allowed deployment types, the use cases it is approved for, the evaluation results that justified it, an owner and a retirement date. The retirement date matters because Microsoft retires model versions on a published schedule. A register without retirement dates turns into a list of surprises.

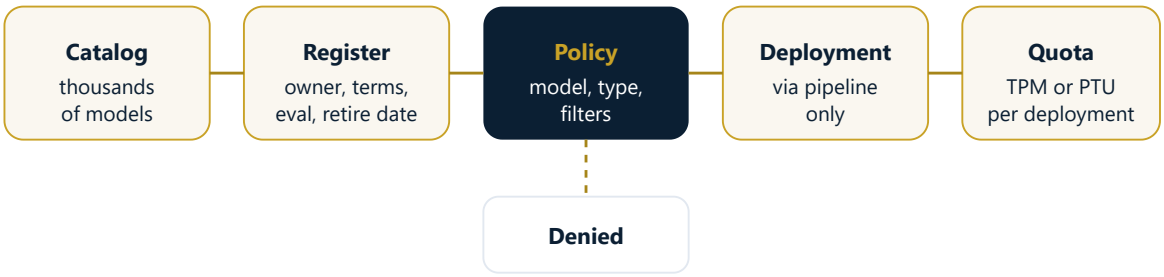
The register becomes a contract when the pipeline turns it into policy parameters. At the time of writing, Azure Policy includes two built-in definitions for this, both supporting Audit and Deny:

- **Foundry model deployments should only use approved models**, which allows deployments only of models you list by identifier or publisher.
- **Foundry model deployments should meet eligibility requirements**, which filters on model attributes such as preview or generally available status and distribution source.

We assign the eligibility policy broadly (for example, no preview models outside Sandbox) and the approved-models policy per environment, with the list generated from the register so there is one source of truth.

FIGURE 4.1

From catalog to a governed deployment



A deployment exists only if the model is on the register, the deployment type is allowed in this environment, and quota is assigned to it by name.

Policy is the gate a person cannot quietly skip. The register says what should happen; policy makes it the only thing that can happen. The one sanctioned way around it is an exemption with an owner and an expiry (Governing Azure by Contract, chapter 6).

Deployment types decide where prompts go

When you deploy a model sold by Azure, you choose a deployment type. It decides three things at once: where prompts and responses are processed, how you pay and how predictable the latency is. At the time of writing, Microsoft documents these types:

Deployment type	SKU name	Processing	Billing
Global Standard	`GlobalStandard`	Any Azure region	Pay per token
Global Provisioned	`GlobalProvisionedManaged`	Any Azure region	Reserved PTU
Global Batch	`GlobalBatch`	Any Azure region	Discounted, asynchronous
Data Zone Standard	`DataZoneStandard`	Within the data zone	Pay per token
Data Zone Provisioned	`DataZoneProvisionedManaged`	Within the data zone	Reserved PTU
Data Zone Batch	`DataZoneBatch`	Within the data zone	Discounted, asynchronous
Standard	`Standard`	Within the Azure geography	Pay per token
Regional Provisioned	`ProvisionedManaged`	Within the Azure geography	Reserved PTU
Developer	`DeveloperTier`	Any Azure region	Fine-tuned model evaluation only; no SLA

Microsoft recommends Global Standard as the default, because new models arrive there first with the broadest availability. That is a sound default for workloads without residency rules. For workloads with them, the residency contract from chapter 2 decides, and policy enforces it. Microsoft documents a policy pattern that matches the deployment SKU. We write it as an allow list, so a deployment type Microsoft adds later is denied until someone approves it. A version for a scope whose contract says "data zone" allows only the Data Zone types. Standard and Regional Provisioned process within the resource's Azure geography, which is a different boundary from a data zone; add them to the list only after the data owner confirms that every allowed region's geography satisfies the residency rule:

```
{
  "mode": "All",
  "policyRule": {
    "if": {
      "allOf": [
        { "field": "type",
          "equals": "Microsoft.CognitiveServices/accounts/deployments" },
        { "field": "Microsoft.CognitiveServices/accounts/deployments/sku.name",
          "notIn": [ "DataZoneStandard", "DataZoneProvisionedManaged",
                     "DataZoneBatch" ] }
      ]
    },
    "then": { "effect": "deny" }
  }
}
```

Fine-tuned models re-enter the register

A fine-tuned model is a new model, not a setting on an old one. It inherits nothing from the base model's approval: its behavior depends on the training data, and that data is now part of what you govern. We treat every fine-tuning job as a register request. The entry records the base model and version, the training and validation datasets with their owner and data classification, where the job ran, and the evaluation run that compared the tuned model against the base model on the same golden set (chapter 7). Training data follows the same rules as grounding data: private storage, managed identity access and no unapproved personal data. Microsoft's Developer deployment type exists for evaluating fine-tuned models, has no SLA and no data residency guarantee, and at the time of writing is deleted after 24 hours, so it belongs in dev only. The production deployment of a tuned model goes through the pipeline like any other, with its own register row and its own retirement date.

Quota is a shared budget, so share it by contract

For standard deployments, quota is measured in tokens per minute (TPM) and assigned to your **subscription**. Its scope is changing. The quota page still describes pools per region, per model and per deployment type, and since May 2026 Microsoft has been moving models to subscription-level pools: Global Standard deployments of one model version share one pool across all regions, and Data Zone Standard shares one pool per data zone. Pipelines should read the scope for each model from the quota APIs rather than assume it. Each deployment takes a slice of that pool when it is created, and the slice becomes its rate limit; requests above it receive HTTP 429 responses. When the pool is used up, the only options are to shrink another deployment or request more. Microsoft notes that increase requests

favor customers who actively use what they already have. Provisioned deployments are measured in provisioned throughput units (PTU) instead and give reserved capacity with more predictable latency.

Three consequences follow. First, the subscription-per-environment split from Book 1 is also a quota split, which is the cleanest way to keep dev from starving prod. Second, inside a shared subscription, TPM per deployment is a decision someone must own; we record it in the register next to the model. Third, a programmatic deployment sets capacity in units whose TPM value varies by model, so pipelines should check the ratio for each model before they assign capacity. Give FinOps and team leads Cognitive Services Usages Reader at subscription scope so they can see the pool, and alert before it runs out.

Not every 429 is a quota problem. Microsoft documents that standard deployments share capacity, and the service may lower a deployment's effective limit for a while under high demand. Check the response headers before you request more quota. For steady production load with tight latency targets, provisioned throughput is the documented answer.

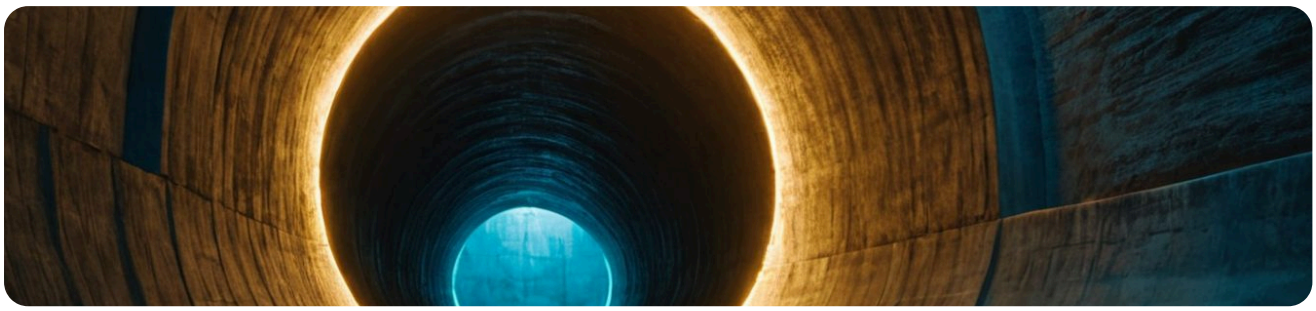
CONTRACT CHECKLIST

- An approved-model register exists with owner, terms, evaluation evidence and retirement date per model version.
- The approved-models and eligibility policies are assigned in every environment, audit first and then Deny.
- Allowed deployment types per environment match the data residency contract and are denied otherwise.
- TPM or PTU per deployment is recorded with an owner, and quota usage alerts reach a person.
- Model retirements on the register are reviewed every month.
- Every fine-tuned model has its own register entry, dataset owner and evaluation run before it is deployed.

See it live: hub.jcardena.com/azure/azure-foundry-catalog.html

CHAPTER FIVE

Private by Default for AI

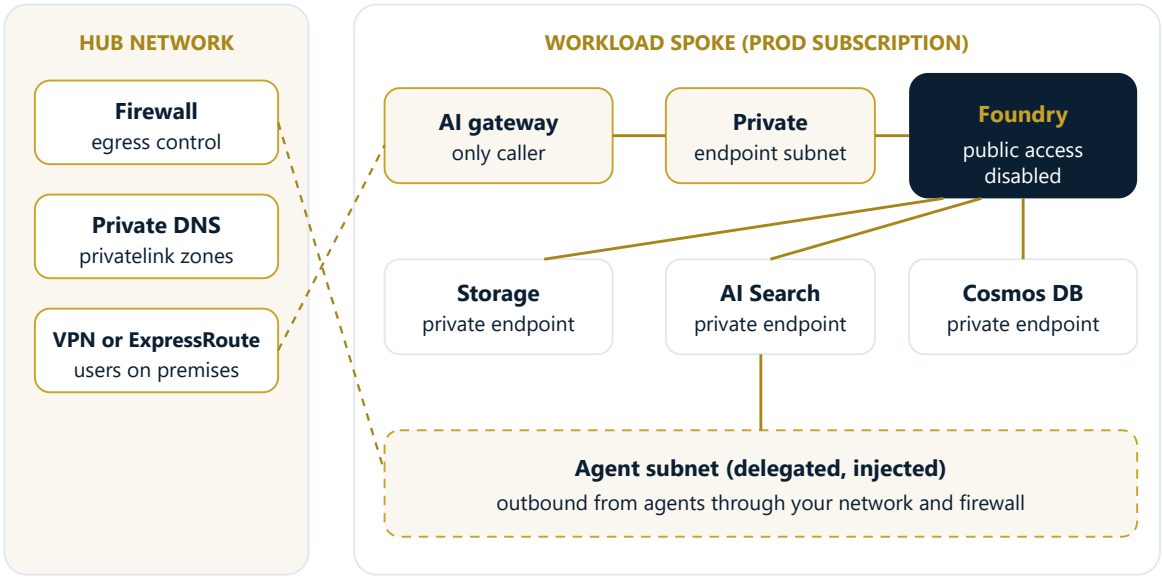


An AI workload moves data in three directions. People and apps send prompts in. The Foundry resource reaches out to your data in storage and search. Agents reach out further, to tools and other services. Each direction needs its own private path, and the third one has a catch: for agents, the outbound network design is chosen when the Foundry resource is created and, at the time of writing, cannot be changed afterward. Decide it on day one.

Governing Azure by Contract, chapter 7, set the network baseline: hub and spoke, a central firewall, private endpoints for data services, private DNS zones in the connectivity subscription and policy that denies public endpoints in production. This chapter applies that baseline to Foundry and points out the places where AI adds new paths.

FIGURE 5.1

Three directions, three private paths



Inbound (top row), resource to data (middle row), agents outbound (bottom row).

Nothing in the spoke has a public endpoint. Users and apps arrive through the hub and the AI gateway, the only caller the Foundry endpoint subnet admits; Foundry features reach data over private endpoints through their documented network path; agent traffic leaves through a delegated subnet and the hub firewall.

Inbound: who can reach the resource

A Foundry resource's `publicNetworkAccess` property is either Enabled or Disabled. The portal shows a third option, selected networks, which is public access Enabled with network rules that deny by default and allow named IP ranges and virtual networks. In production we set it to **disabled** and add a private endpoint in the workload spoke. Users reach the portal and APIs through the hub, over VPN, ExpressRoute or a jump host through Azure Bastion. Microsoft notes that removing a private endpoint does not make a resource public, and that enabling public access does not remove private endpoints, so check both settings during reviews.

Disabling public access is not the same as routing everything through the AI gateway of chapter 9. Any workload with a network path to the private endpoint and a role on the resource can still call Foundry directly. In production we make the gateway the only path with three controls that each work on their own: the Foundry private endpoint sits in a dedicated subnet whose network security group (with private endpoint network policies enabled) and firewall routes admit only the gateway subnet; only the gateway's managed identity holds an inference role for application traffic on the production resource, with the

platform exceptions listed in chapter 9; and key access is disabled, so a caller without that role has nothing to present. Applications in the spoke resolve and reach the gateway, never the Foundry endpoint.

Private endpoints only work if names resolve to private addresses. At the time of writing, Microsoft's agent networking guidance lists three zones for the Foundry resource itself: `privatelink.cognitiveservices.azure.com`, `privatelink.openai.azure.com` and `privatelink.services.ai.azure.com`. A network-isolated agent setup also needs the zones for the services it depends on: `privatelink.blob.core.windows.net`, `privatelink.search.windows.net` and `privatelink.documents.azure.com` (the zone for Azure Cosmos DB for NoSQL, which the agent setup uses). Keep all of them in the connectivity subscription, linked to the hub, as Book 1 does for every other service.

Resource to data: no public data paths

Foundry is only as private as the data it reads. Storage accounts, Azure AI Search and Azure Cosmos DB each get their own private endpoint, and Microsoft notes that these are not created for you when you deploy the Foundry resource. Access goes through managed identities with data roles, never keys. Some data services also offer a trusted-services exception that lets named Azure services reach them with their managed identity. It is a firewall bypass on a public endpoint that stays enabled, not a private path, and support differs by service and feature. Treat it as a written exception with an owner, not as part of the no-public-path guarantee. A private endpoint on the data service is also only half of the path: the Foundry feature that reads the data (agents, evaluations, indexers, project connections) needs its own documented way into the network, such as the managed virtual network or a delegated agent subnet. Check that path feature by feature before you promise it.

For agents, Microsoft's **standard setup** brings your own storage, search and Cosmos DB, so all agent data stays in your tenant, in resources you govern. The **basic setup** uses platform-managed resources. For regulated workloads we use the standard setup only.

Private paths and encryption keys are separate decisions. Data at rest is encrypted with Microsoft-managed keys by default; where regulation or the data owner requires it, the contract asks for customer-managed keys in Azure Key Vault for the Foundry resource and for each dependency that stores AI data: storage, AI Search and Cosmos DB. At the time of writing, Azure Policy includes *Azure AI Services resources should encrypt data at rest with a customer-managed key (CMK)*, with Audit and Deny. Support and limits differ by service and feature, so check each one, and give the key vault the same private endpoint, purge protection and PIM-only access as any other production secret store.

Private paths decide which services can read the data. They do not decide which user may see which document. A retrieval system that indexes a whole document library under one service identity will happily ground an answer for a junior analyst on a board paper. The contract needs document-level security trimming: every query carries the caller's identity, and the index returns only documents that identity may read. At the time of writing, Azure AI Search documents a generally available security filter pattern, where the application filters on group identifiers stored with each document, and native document-level access control based on Microsoft Entra tokens, ACLs and sensitivity labels, in preview for specific data sources. Choose one per use case, test it with a user who must not see a document, and add that case to the golden set (chapter 7).

> **The silent empty index.** An Azure AI Search indexer that must reach data through private endpoints needs its execution environment set to private. Microsoft documents that otherwise it runs in the multitenant environment, which cannot use the private endpoint. The indexer run then fails, or leaves an empty or stale index, and nobody notices unless indexer status and document counts are monitored. An agent grounded on an empty index does not fail. It answers from nothing.

Agents outbound: decide at birth

Agents call tools, and tools are outbound traffic. Microsoft offers two ways to keep that traffic inside your boundary. With **virtual network injection**, you bring your own virtual network: the agent client runs in a subnet delegated to `Microsoft.App/environments`, sized /27 or larger at the time of writing, and its traffic goes through your routes and your firewall. A **managed virtual network**, created and run by Microsoft for the resource, is also documented, with parts of it in preview at the time of writing. We use injection for production so that egress passes the same firewall and logs as everything else.

The design choice is permanent. At the time of writing, Microsoft documents that you cannot add outbound injection to an existing Foundry resource or move it to a different subnet; you must redeploy. That is why chapter 1 warned against piloting outside the landing zone.

Even with injection, not every tool stays private. Microsoft's tool support table, at the time of writing, shows private MCP servers, OpenAPI tools, Azure Functions and AI Search going through your subnet or a private endpoint, while Bing grounding, web search and SharePoint grounding still use public endpoints, and some tools are not yet supported in isolated setups at all. Decide per environment which tools are allowed, and enforce it with the preview built-in policy *Microsoft Foundry agents and toolboxes should only use approved tools*, or its companion *should not use denied tools*; at the time of writing both support Audit and Deny.

> **THE PUBLIC GATEWAY**
>
> An AI gateway added from the portal in front of a private Foundry resource does not inherit its privacy. At the time of writing, Microsoft documents that a gateway created this way is public, and must be isolated separately through API Management networking. Chapter 9 covers the gateway; the rule here is that every new component in front of a private resource gets its own network review.

Policy makes it the default

Built-in policy (name at the time of writing)	First phase, every environment	Once clean, every environment
Azure AI Services resources should restrict network access	Audit	Deny
Configure Cognitive Services accounts to disable public network access	A Modify policy, deployed with its effect set to Disabled while remediation is tested	Modify
Azure AI Services resources should use Azure Private Link	Audit	Audit (it offers no Deny effect)
Microsoft Foundry agents and toolboxes should only use approved tools (preview)	Audit	Deny

The same guardrails apply in dev, test and prod, as in Governing Azure by Contract, chapter 6. What changes between environments is the parameter list, such as allowed tools, not the effect. Sandbox subscriptions, under the Sandbox management group of Governing Azure by Contract, chapter 3, keep the looser first-phase set.

CONTRACT CHECKLIST

- Foundry resources outside Sandbox have public network access disabled; in production the private endpoint admits only the gateway subnet.
- All required privatelink DNS zones live in the connectivity subscription and resolve from the spoke.
- Storage, AI Search and Cosmos DB used by Foundry have their own private endpoints and managed identity access.
- The outbound design for agents (injection or managed network) is decided before the resource is created.
- Allowed agent tools per environment are written down and enforced by policy.
- Every component added in front of Foundry, including the gateway, has its own network review.

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

CHAPTER SIX

Safety and Responsible AI



A model will say things you did not plan for. Safety on Microsoft Foundry is the set of checks that sit between the model and the people it serves: what may go in, what may come out, and what an agent may send to its tools. These checks are not a promise the model makes. They are a contract the platform runs on every call, and like every contract in this book, each one needs an owner, a setting per environment and a record that shows it ran.

The first five chapters put Foundry inside the landing zone, gave people and workloads the right roles, put models behind an approved register and quota, and closed the public doors. All of that controls who can reach a model. This chapter controls what the model is allowed to do once it is reached. It opens the guarantee chapters of Part II and sets up the next one: safety settings say what must never happen, and evaluations in chapter 7 prove how often it still does.

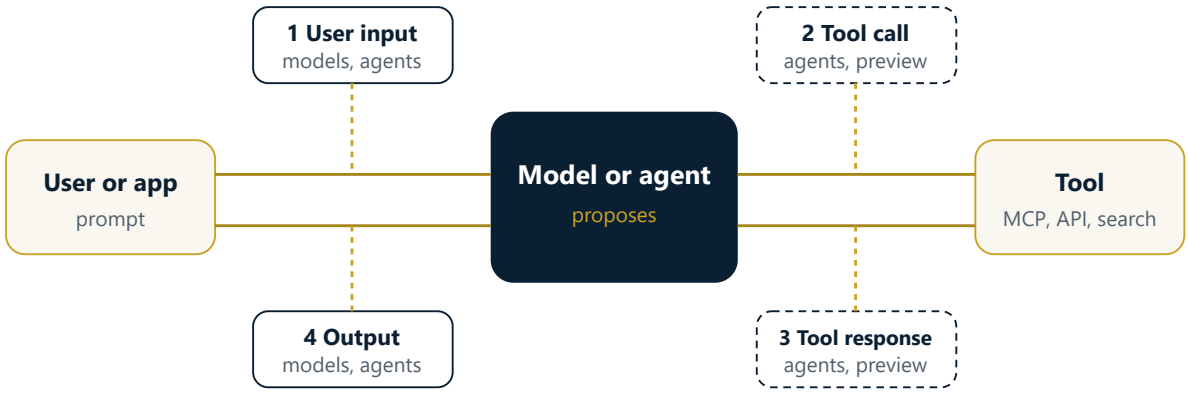
Guardrails: a named set of controls

Microsoft has changed the vocabulary here, so start with the current terms. In the Foundry documentation a **guardrail** is a named collection of **controls**. Each control names three things: a risk to detect, an intervention point where the platform looks for it, and the action to take when it finds it. Older pages and the Foundry (classic) portal call the same configuration a *content filter*, and the REST API still calls it an RAI policy. The detection itself comes from Azure AI Content Safety classification models.

There are four intervention points. **User input** is the prompt sent to a model or agent. **Output** is the completion returned to the user. For agents only, there are two more: **tool call**, what the agent proposes to send to a tool, and **tool response**, what comes back from it. Agent guardrails, including the tool call and tool response points, are in preview at the time of writing.

FIGURE 6.1

Where the guardrail looks



Each control = risk + intervention point + action (annotate, or annotate and block)

Four doors around every call. Models get two checkpoints. Agents get four, because what an agent sends to a tool and what it reads back are doors too. Dashed boxes are in preview at the time of writing.

What the controls detect

The risks fall into three families. Put them in your contract by family, so the contract still reads correctly when a product page changes its labels.

Family	Risks (current names)	What it protects
Harmful content	Hate, Sexual, Violence, Self-harm, each classified by severity (safe, low, medium, high)	People who read the output, and the brand
Attacks	Prompt Shields: user prompt attacks (jailbreaks) and indirect attacks (instructions hidden in documents or tool results); spotlighting, in preview	The system's own instructions and data
Output integrity	Protected material for text and code; personally identifiable information (preview); groundedness (preview); task adherence (preview)	Rights holders, privacy, and the truth of the answer

Two details matter for planning. First, groundedness detection in Azure OpenAI is in preview and Microsoft has announced its retirement on April 30, 2027; Microsoft points customers to evaluation tooling or their own grounding checks instead. That is one reason chapter 7 treats groundedness as an evaluation obligation, not only a filter. Second, indirect

attack detection only works when the application marks retrieved documents with the document delimiters Microsoft describes. A filter that cannot see where the document starts cannot tell an instruction from data.

Defaults, overrides and who may change them

Every model deployment gets a default guardrail, currently named **Microsoft.DefaultV2**, and the Microsoft default guardrails cannot be edited. To change behavior, a team creates its own guardrail and assigns it. For agents the rule is precise: an agent with its own guardrail uses that guardrail, and it fully overrides the guardrail of the model underneath. An agent without one inherits the guardrail of its model deployment. Write that rule into your contract, because it means an agent can be looser than the model it calls if nobody checks.

Teams can tighten thresholds freely. Turning off filtering on outputs for Microsoft-sold models requires approval through Microsoft's limited access review for modified guardrails. Our platform team adds its own layer on top of that: no project changes a guardrail in test or production without a pull request, a named approver from the responsible AI owner, and an entry in the evidence store. The guardrail is configuration, so it belongs in the same infrastructure as code pipeline as everything else (*Governing Azure by Contract*, chapter 11).

> **Know how it fails.** Microsoft documents that if the filtering system is unavailable or cannot finish in time, the request still completes without filtering, and the response carries an error inside `content_filter_results`. That is fail-open behavior. Your application contract must say what happens then: check for that error on every response, and for high-risk flows, hold or block the answer instead of showing it.

A safety tier per use case

Not every AI use case needs the same settings. An internal search assistant over policy documents and a public assistant that can take actions do not carry the same risk. We set three safety tiers. Every project declares its tier at intake, and the approved-model register from chapter 4 records which tiers each model may serve.

Tier	Example	Guardrail floor	Human step
T1 Inform	Internal Q&A over approved documents	Default or stricter; Prompt Shields on input; protected material on output	None per answer; sampled review
T2 Advise	Drafts that a person sends or signs	Custom guardrail; indirect attack detection with document delimiters; PII control where data allows	Person approves before anything leaves

T3 Act	Agent that writes data or calls external systems	Agent guardrail on all four intervention points; tool allowlist	Approval on every write or external call (chapter 10)

Human approval is a control too

Classifiers find harmful words. They do not know whether a refund is too large or whether an email should go to that customer. For actions that cannot be undone, the strongest control is still a person who sees the proposed action and says yes. Foundry supports this for agents that call MCP tools: a tool can be configured to require approval, and the agent pauses until the application returns an approval response. Chapter 10 builds the agent side of this. For this chapter, the rule is simple: the safety contract names which actions need a person, and the platform enforces the pause.

> **A guardrail is not the model being careful.** *It is the platform refusing to pass what the contract forbids, whether or not the model was careful.*

Defense in depth at the gateway

Foundry guardrails apply to models sold by Azure and to agents built in Foundry Agent Service. They do not cover every agent a team registers from elsewhere, and they do not cover models called outside Foundry. For that traffic, Azure API Management offers an `llm-content-safety` policy that sends prompts to Azure AI Content Safety and blocks on a threshold; completions are checked too when the policy sets `enforce-on-completions` or is placed in the outbound section. Chapter 9 places this policy in the AI gateway, so every model call in the estate passes at least one safety check, wherever the model lives.

> **THE CLAIMS ASSISTANT**
>
> A claims team wanted an assistant that drafts replies to customers from case notes. At intake it declared tier T2. The platform team assigned a custom guardrail with Prompt Shields for both user and indirect attacks, wrapped the case notes in document delimiters so injected text in an attachment could be detected, and enabled the PII control on output. The reply goes to a handler who edits and sends it. Nothing reaches a customer without a person, and every blocked request is logged with the control that blocked it.

CONTRACT CHECKLIST

- Every project has a declared safety tier, recorded in the approved-model register.
- Every deployment and agent has an explicitly assigned guardrail; no agent relies on silent inheritance in production.
- Guardrail changes in test and production go through a pull request with a named responsible AI approver.
- Applications check `content_filter_results` for errors and hold high-risk answers when filtering did not run.
- Retrieved documents are marked with document delimiters wherever indirect attack detection is required.
- Actions that cannot be undone require human approval, enforced by the platform, not by the prompt.

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

CHAPTER SEVEN

Every Answer Is Earned: Evaluations



An AI system does not pass or fail once. It is good enough some of the time, and the question is how often, measured how, and who decides that the number is good enough to ship. Evaluations turn "it seemed fine in the demo" into a number with a threshold, a test set and an owner. In a governed estate, no model, prompt or agent moves from dev to test, or from test to production, without an evaluation gate that it passed and a record that proves it.

Chapter 6 set what must never happen. This chapter measures how close the system gets to what must happen. In the language of the GDCD series, evaluations are how the contract's *tolerances* clause is enforced: the clause names the metric, the sample, the threshold and the action on breach, and the evaluation run is the evidence. Every answer is earned.

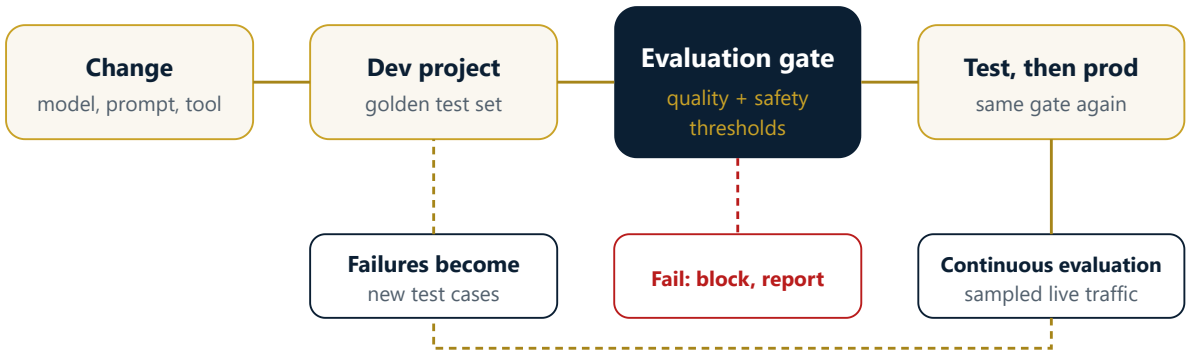
Three kinds of evaluation, one gate

We run three kinds of evaluation, at three points in the life of a change.

- **Pre-release evaluation** runs a fixed test set against the candidate in the pipeline. It blocks promotion if a threshold is missed. This is the regression gate.
- **Adversarial testing** sends attack prompts and unsafe requests on purpose, to see what gets through the guardrails from chapter 6. Foundry offers scheduled red team scans for agents, in preview at the time of writing.
- **Continuous evaluation** scores a sample of live production traffic, so drift shows up as a falling line, not as a complaint.

FIGURE 7.1

The evaluation gate in the promotion path



Every production failure is a test case the next release must pass.

One gate, run at every promotion. The loop at the bottom is what makes the test set better over time: what slipped through in production is added to the golden set.

The built-in evaluators

Foundry ships a catalog of built-in evaluators. Some are model-based judges, some use Azure AI Content Safety, and some are plain math. Names change, so the contract refers to families and the pipeline pins the exact evaluator names. At the time of writing the families are these.

Family	Examples	Use it for
General purpose	Coherence, Fluency	Readability of any text answer
RAG	Retrieval, Groundedness, Relevance; Groundedness Pro and Response Completeness in preview	Answers built from retrieved documents
Risk and safety	Hate and Unfairness, Sexual, Violence, Self-Harm, Protected Materials, Indirect Attack, Code Vulnerability, Prohibited Actions, Sensitive Data Leakage	The safety floor for every release
Agent	Tool Call Accuracy, Tool Selection, Tool Input Accuracy, Tool Call Success; Task Adherence, Task Completion and Intent Resolution in preview	Whether an agent chose and used tools correctly
Textual similarity	F1, BLEU, ROUGE, METEOR, Similarity	Answers with a known expected text
Custom	Rubric evaluators and custom evaluators, in preview	Business rules no built-in evaluator knows

Two practical points. Model-based evaluators need a judge model deployment, which uses quota from chapter 4; give the evaluation workload its own deployment so it never competes with production. And a built-in evaluator measures a general quality. Your business outcome is specific. Most mature projects end up with one or two custom evaluators that check the thing the business actually cares about, such as "the cited policy number exists" or "the refund amount is within the agent's limit".

The test set is the contract

An evaluation is only as good as its test set. We call it the **golden set** and treat it as a governed asset: versioned in the repository, owned by the business owner of the use case, reviewed when the use case changes. A useful golden set has four parts: typical questions with expected answers, edge cases the team knows are hard, safety cases that must be refused, and every real failure found in production since the last release. Keep it free of real customer data unless the data owner has approved it, because the test set is copied into pipelines and reports.

> **Do not let the system grade itself.** If the same model writes the answer and judges it, a shared blind spot passes twice. Use a different judge deployment, keep the judge prompt under change control, and calibrate the judge against a small human-graded sample before trusting its scores.

Running the gate in the pipeline

Foundry gives two SDK paths. The Azure AI Evaluation SDK (the `azure-ai-evaluation` Python package) runs evaluators locally, which suits quick checks during development. Cloud evaluations run in the Foundry project through the Foundry project SDK (`azure-ai-projects`, version 2 or later in the current documentation) and store results in the project, which is what the gate needs as evidence. The pipeline from *Governing Azure by Contract*, chapter 11, runs the cloud evaluation with its federated workload identity, reads the scores, compares them to thresholds kept in the repository, and fails the stage if any threshold is missed.

```
# eval-gate.yaml (kept in the repository, reviewed like code)
use_case: claims-reply-assistant
golden_set: tests/golden/claims-v7.jsonl
judge_deployment: eval-judge-prod
thresholds:
  groundedness_pass_rate: 0.95     # share of answers judged grounded
  relevance_pass_rate: 0.90
  safety_defects_allowed: 0        # any hate/sexual/violence/self-harm defect fails
  indirect_attack_defects_allowed: 0
on_fail: block_promotion, open_ticket(owner=claims-ai-owner)
```

The numbers in this file are examples for one use case, not Microsoft guidance. The point is that they are written down before the run, by the owner, and the pipeline applies them without negotiation. A threshold argued after the results is not a threshold.

Continuous evaluation in production

Pre-release tests cover what you thought of. Production shows you what you did not. Foundry supports two kinds of recurring evaluation for agents: **scheduled evaluations** that run on a fixed schedule, and **continuous evaluation** that samples live traffic as responses complete. Continuous evaluation is configured as an evaluation rule with a cap on runs per hour, so cost stays bounded. Results appear in the agent's Monitor tab and in the Application Insights resource connected to the project. The project's managed identity needs the **Foundry User** role on the Foundry resource for continuous evaluation rules (Microsoft recently renamed the Azure AI roles; Foundry User was Azure AI User). Recurring evaluations and evaluation alerts are in preview at the time of writing. Chapter 8 connects these scores to traces, so a falling score leads straight to the conversations that caused it.

> **A MODEL UPGRADE THAT DID NOT SHIP**
>
> A team wanted to move a document assistant to a newer model version. The new model was faster and scored higher on fluency. The gate ran the golden set and groundedness fell below the threshold on a group of policy questions with long tables. The promotion was blocked automatically. The team adjusted retrieval chunking, added the failing questions to the golden set, and the next run passed. No one had to argue about it in a meeting.

CONTRACT CHECKLIST

- Every use case has a versioned golden set owned by its business owner.
- Thresholds for quality and safety are written in the repository before evaluation runs.
- Promotion from dev to test and from test to production is blocked by the pipeline when any threshold is missed.
- The judge model is a separate deployment, calibrated against human-graded samples.
- Production traffic is sampled by continuous evaluation, and every confirmed failure is added to the golden set.

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

PART III · RUN

CHAPTER EIGHT

Tracing and Visibility



When an AI answer goes wrong, the first question is always the same: what happened? Who asked, which model answered, which documents it read, which tools it called, how long each step took, and what it cost. If you cannot answer that within minutes, you are not running an AI system. You are watching one. Tracing is the record that lets you answer, and on Microsoft Foundry it is built on OpenTelemetry and stored in Azure Monitor Application Insights.

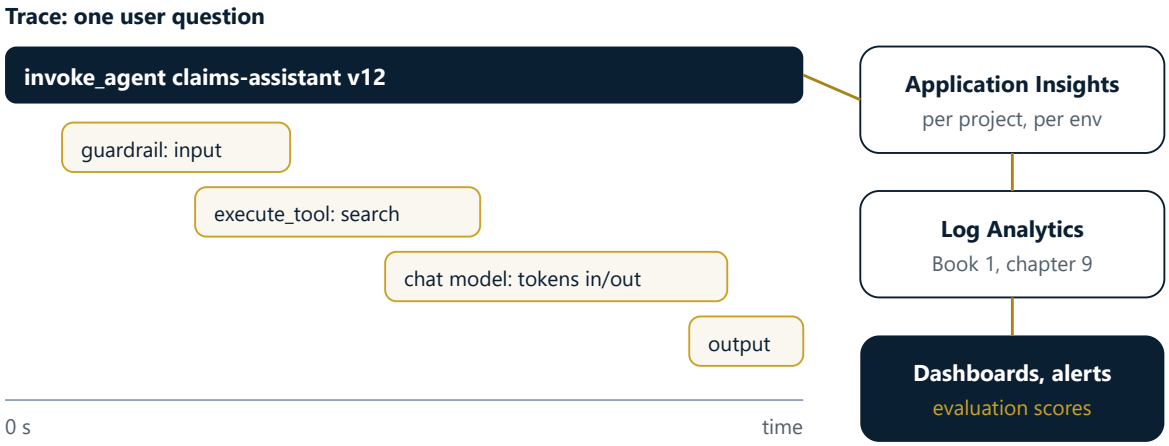
Part II built the system: safe settings in chapter 6 and an evaluation gate in chapter 7. Part III is about running it. We start with visibility, because every later control depends on it. The token limits in chapter 9 need usage data. The agent approvals in chapter 10 need a record of what was approved. And the evaluation scores from chapter 7 only become useful when you can click from a falling score to the conversations behind it.

What a trace is

A **trace** is the full journey of one request. It is made of **spans**, one per operation, nested to show which step called which. Each span carries **attributes**: the model name, token counts, the tool called and its arguments, the outcome. OpenTelemetry defines semantic conventions for generative AI so these attributes have the same names across frameworks, with span names such as `invoke_agent` and `execute_tool`. Microsoft notes that the GenAI conventions are still in Development status in OpenTelemetry and may change, so pin the versions your queries depend on.

FIGURE 8.1

One request, one trace, one place to look



Span attributes answer: who, which model, how long, how much.

The waterfall is the evidence. Each bar is a span. Read across for latency, down for causality. The same records feed the dashboards and the evaluation scores, so there is one version of what happened.

How Foundry collects traces

Tracing on Foundry starts by connecting an Application Insights resource to the project. Once connected, Foundry captures traces for agents it runs. At the time of writing, Microsoft lists tracing as generally available for prompt agents and hosted agents, and in preview for workflow agents and external agents. For your own application code, and for agents built with frameworks such as Microsoft Agent Framework, LangChain, LangGraph or the OpenAI Agents SDK, you add client-side instrumentation that exports OpenTelemetry spans to the same Application Insights resource. Agents that run outside Foundry can be registered in the Foundry control plane and send telemetry to the same resource, so they appear in the same views.

Where those resources live is a landing zone decision, not a project decision. Our standard is that each Foundry project environment has its own workspace-based Application Insights resource, deployed by the platform's infrastructure as code, sending to the environment's Log Analytics workspace from *Governing Azure by Contract*, chapter 9. Dev traces never land in the production workspace, and production traces never land in a workspace a developer can read by default.

Traces hold sensitive data

A trace can contain the user's question, the model's answer, the documents retrieved and the arguments sent to tools. That is often more sensitive than anything else in your logs. Microsoft's guidance is direct: never put secrets or credentials in prompts, tool arguments or span attributes; redact or minimize personal data before it reaches telemetry; and treat traces as production telemetry with the same access controls and retention as other logs. Foundry also documents a way to route sensitive content to a dedicated table and restrict who can read it.

> **Two audiences, two levels of access.** Operators need latency, errors and token counts. Very few people need the content of prompts and answers. Split them: metrics and span timing readable by the operations group, message content restricted to a small investigation group through PIM, with every read logged.

Prompts and traces are records

Once prompts, answers and traces are stored, they are business records with a classification, a retention period and, sometimes, a legal hold. Write those three into the contract before go-live. Set log retention per table in Log Analytics to match the records schedule, keep content tables shorter than metric tables when the schedule allows, and make deletion a tested procedure, not a request to an engineer. Microsoft Purview adds the governance layer that logs alone do not give you. At the time of writing, Microsoft documents a Foundry integration with Purview that records prompts and responses in the Purview audit log, shows them in Data Security Posture Management for AI, applies retention policies through the Enterprise AI apps location, supports eDiscovery, and applies Purview data security policies to calls made with Entra user-context authentication. Microsoft also notes limits, such as no support yet for network-isolated setups, so check the current page against your design. Sensitivity labels on grounding data should travel with it into the index (chapter 5). Telemetry is not the only store: uploaded files, vector stores, conversation threads, agent memory and fine-tuning datasets each need an owner, a retention period and a tested delete, and in the standard agent setup they live in storage, search and Cosmos DB resources you govern.

Who asked, which model answered

The Book 2 promise is that every project, model, key and token has an owner, a limit and a log. Tracing is the log, and it only works if each request carries the right identifiers. We require four on every trace:

1. **Caller**: the application or agent identity, and a pseudonymous user ID from the application, never a raw email address.
2. **Use case and version**: the agent name and version, or the application release.
3. **Model and deployment**: the deployment name, which maps to the approved-model register from chapter 4.
4. **Contract version**: which guardrail and which evaluation gate were in force.

With these in place, the questions that come up in a review can be answered by a query instead of a meeting. The query below is an example; the attribute names follow the OpenTelemetry GenAI conventions, so confirm them against your own data before you build alerts on them. It runs in Application Insights; in the Log Analytics workspace the same records sit in the `AppDependencies` table, with `TimeGenerated`, `Properties` and `DurationMs` in place of the column names below.

```
dependencies
| where timestamp > ago(1d)
| where isnotempty(customDimensions["gen_ai.operation.name"])
| extend model = tostring(customDimensions["gen_ai.request.model"])
| summarize calls = count(),
            p95_ms = percentile(duration, 95),
            failures = countif(success == false),
            tokens_in = sum(toint(customDimensions["gen_ai.usage.input_tokens"])),
            tokens_out = sum(toint(customDimensions["gen_ai.usage.output_tokens"]))
          by model, bin(timestamp, 1h)
```

The dashboards that matter

Foundry includes an Agent Monitoring Dashboard on each agent's Monitor tab, reading from the connected Application Insights resource. It shows token usage, latency, run success rate, evaluation scores from chapter 7, and red team results where enabled. Several of its features, including the metric views and alerts, are in preview at the time of writing. We use it for the team that owns the agent, and add one estate-level workbook for the platform team, built on Log Analytics, with the same four views for every project.

View	Signal	Who acts
Health	Failure rate, latency percentiles, throttling (429) responses	Workload on-call
Quality	Continuous evaluation scores, trend against the release baseline	Use case owner
Safety	Blocked requests by control, filter errors, red team findings	Responsible AI owner
Spend	Tokens by project, deployment and caller (with chapter 9 gateway metrics)	FinOps lead

Write the targets for these views into the contract as one service level line per production use case, for example: quality pass rate on sampled traffic at or above the release baseline, zero tolerated high-severity safety defects, a p95 latency, and an availability target measured at the gateway. The numbers belong to the business owner; the line makes them testable.

An alert only counts if it reaches a person who can act. Route each view's alerts to the role in the last column, through the action groups set up in *Governing Azure by Contract*, chapter 9. Agent identities also show up in Microsoft Entra sign-in and audit logs, which chapter 10 uses to see which agent obtained a token for which resource.

Detect threats and drift

Everything so far is preventive or diagnostic. A CISO also needs a detective control that runs whether or not a team remembered its checklist. That is Microsoft Defender for Cloud, already in place from Governing Azure by Contract, chapter 8. Two parts matter for AI. **AI security posture management**, part of the Defender CSPM plan, discovers AI resources across subscriptions, including Foundry accounts and projects, and raises recommendations and attack paths for gaps such as public exposure or weak identity settings. **AI threat protection**, the Defender for AI Services plan, raises alerts for threats against model deployments, such as jailbreak attempts detected by Prompt Shields, and sends them to the Microsoft Defender portal and to Microsoft Sentinel. At the time of writing, Microsoft has moved agent-level discovery, posture and threat protection for Foundry agents to a Microsoft Agent 365 license, while the Defender for AI Services plan keeps covering models. Check the licensing page before you design the agent part of the control. Whatever the license, the contract is the same: the plans are on for every subscription that holds Foundry, every alert has an owner in security operations, and a finding closes with evidence, not a comment.

> **THE SLOW MONDAY**
>
> Latency on an internal assistant doubled every Monday morning. The trace waterfall showed the model call was fast; the search tool span was slow, and only for one index that a weekly job rebuilt at the same hour. The fix was a schedule change, not a bigger model. Without spans per tool, the team would have bought capacity it did not need.

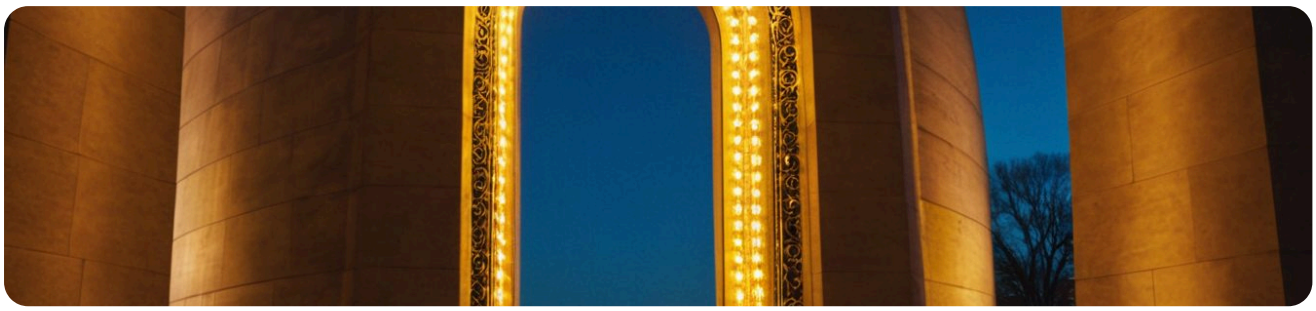
CONTRACT CHECKLIST

- Every Foundry project environment has its own Application Insights resource, deployed as code and linked to that environment's Log Analytics workspace.
- Every trace carries caller, use case and version, deployment, and contract version.
- No secrets or raw personal data appear in prompts, tool arguments or span attributes; content access is restricted and logged.
- Health, quality, safety and spend views exist for every production project, each with an alert that reaches a named role.
- Retention, classification and deletion for prompts and traces follow the records schedule, with Purview audit and retention configured where the design supports it.
- Defender AI posture and threat protection are on for every Foundry subscription, and every alert has an owner in security operations.

See it live: hub.jcardena.com/azure/azure-ai-modules.html

CHAPTER NINE

Tokens Are Spend: The AI Gateway



Every token a model reads or writes is money, and quota is shared. Without a control point, the first team to ship a busy application uses the capacity that three other teams were counting on, and the invoice arrives with no way to say who spent what. An AI gateway is the single door that all model traffic passes through. It gives each team a token budget, records who used how much, spreads load across deployments and regions, and keeps one audit trail. Every number adds up, because every request is counted at one door and reconciled to the invoice.

Chapter 4 set quota on the deployments themselves. That works while one application calls one deployment. As soon as several applications share deployments, or one application calls deployments in several regions, quota on the deployment cannot say which team used it. The gateway can. In the estates we govern, the gateway is Azure API Management, using the set of capabilities Microsoft calls the **AI gateway in Azure API Management**. It is part of the API Management gateway, not a separate product.

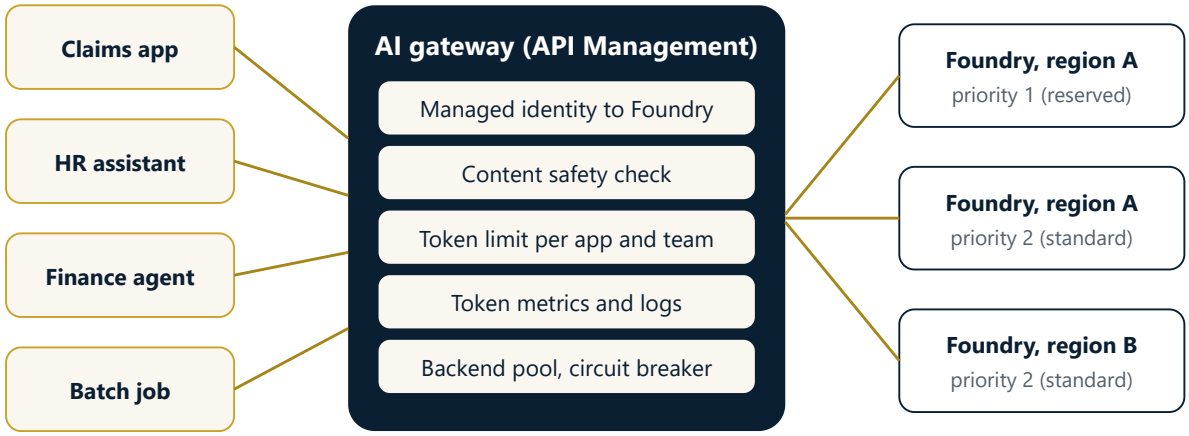
Where the gateway sits

The gateway lives in the connectivity or shared services layer of the landing zone, owned by the platform team and reachable only on the private network from chapter 5, which needs an API Management tier that supports your chosen private networking option. Applications and agents call the gateway; the gateway calls Foundry with its own managed identity, so no application holds a model key. Callers authenticate with Microsoft Entra ID: the `validate-azure-ad-token` policy checks each token and the allowed client application IDs before any other policy runs. Each team gets an API Management product and each application a subscription under it. The subscription is a metering key for budgets, not proof of identity; the validated token is the identity.

The "only path" has documented exceptions, and the contract lists them. The project's managed identity holds Foundry User for continuous evaluation (chapter 7), and agents in Foundry Agent Service call model deployments inside the account, not through the gateway, unless an agent is configured to use a model through a gateway connection. Neither passes the gateway's token limits, so their spend is capped another way: dedicated deployments with their own TPM recorded in the register, and the Foundry-associated gateway described below where it applies. Anything else that reaches the Foundry endpoint is a defect.

FIGURE 9.1

One door for every model call



No application holds a model key. Every token is counted once, at one door.

The gateway is the cost layer's enforcement point. Applications on the left only know the gateway. The platform team can add, move or replace deployments on the right without changing any application.

Token budgets per team

The `llm-token-limit` policy limits tokens per counter key, as a rate (tokens per minute), a quota over a period (hourly to yearly), or both. A caller over the rate gets HTTP 429; over the quota, HTTP 403. The counter key can be any policy expression. Set every AI API to require a subscription, so both keys below always have a value. Older pages use names such as `azure-openai-token-limit`; use the `llm-` policies for new work.

We apply a rate per application, so one noisy app cannot starve the others, and a monthly token ceiling per team. The ceiling is a consumption limit, not a currency budget; the currency view comes from Cost Management. The values below are examples; real ones come from the FinOps review in *Governing Azure by Contract*, chapter 10.

```
<inbound>
    <base />
    <!-- Rate per application (APIM subscription) -->
    <llm-token-limit counter-key="@(context.Subscription.Id)"
        tokens-per-minute="20000"
        estimate-prompt-tokens="false" />
    <!-- Monthly budget per team (APIM product) -->
    <llm-token-limit counter-key="@(context.Product.Id)"
        token-quota="50000000" token-quota-period="Monthly"
        estimate-prompt-tokens="false"
        remaining-quota-tokens-header-name="x-team-tokens-remaining" />
    <!-- Usage metrics for showback -->
    <llm-emit-token-metric namespace="ai-gateway">
        <dimension name="Product ID" />
        <dimension name="Subscription ID" />
        <dimension name="API ID" />
    </llm-emit-token-metric>
</inbound>
```

> **Know what the counter counts.** Microsoft documents that the policy counts prompt and completion tokens, estimates tokens when responses are streamed, can be briefly exceeded by concurrent requests, and keeps its count per gateway, not across regional gateways. Treat the quota as a strong guardrail, not as an exact meter, and reconcile against Cost Management each month.

Showback and chargeback

The `llm-emit-token-metric` policy sends token counts to Application Insights as custom metrics. API Management allows up to five custom dimensions per policy, so pick what the finance conversation needs: team, application and API are usually enough. Turn on LLM logging in API Management for audit, with the content access split from chapter 8. The conversation then moves from "why is the bill high" to "which use case grew, and is it worth it".

The invoice is in currency, and tokens are only part of it. A FinOps-ready report adds four lines. **Provisioned throughput** is paid whether used or not, so report utilization next to cost and reserve only the base load that utilization proves. **Partner models** bill through Azure Marketplace (chapter 4), not on Azure meters. **The platform around the model**, from API Management and AI Search to Content Safety calls and Log Analytics ingestion, scales with AI traffic. **Allocation** uses the tags from Governing Azure by Contract, chapter 10, budgets and anomaly alerts per AI subscription, and the token metrics above to split shared PTU or the gateway. The unit the business needs is cost per outcome, such as cost per case handled.

The contract design also has commercial dependencies, and the business case should name each one and its owner: the API Management tier (private networking and multi-region drive it), Defender CSPM and Defender for AI Services, a Microsoft Agent 365 license for agent-level security (chapter 8), Purview, Application Insights and Log Analytics ingestion and retention, Content Safety calls, and any PTU reservation. Each is a line on the bill, and each control in this book that depends on one fails quietly if the license lapses.

Failover and load balancing

API Management backends can be grouped into a pool with round-robin, weighted or priority-based distribution; at the time of writing Microsoft documents up to 30 backends per pool. Each backend can have a circuit breaker that honors the `Retry-After` header, and lower priority backends are used only when every backend in the higher priority group has tripped. That gives a clean pattern: reserved capacity first, standard deployments second, a second region last.

```
resource pool 'Microsoft.ApiManagement/service/backends@2023-09-01-preview' = {
  parent: apim
  name: 'foundry-chat-pool'
  properties: {
    description: 'Reserved first, then standard, then second region'
    type: 'Pool'
    pool: {
      services: [
        { id: ptuBackend.id, priority: 1, weight: 1 }
        { id: stdBackendA.id, priority: 2, weight: 1 }
        { id: stdBackendB.id, priority: 2, weight: 1 }
      ]
    }
  }
}
```

The API version is the one Microsoft's pool sample uses at the time of writing; pin the version your pipeline validates. Failover to another region moves processing there, so the pool lists only regions the residency rules allow (chapter 2). Microsoft also notes that a throttled model service can return a very long `Retry-After`, so test the circuit breaker before production depends on it.

Continuity when a region or a quota fails

Write continuity as a contract with a recovery time and point per use case, and name the failure domain of each fallback. **Regional outage**: Foundry does not fail over on its own; Microsoft's resiliency guidance describes a second Foundry resource in another allowed region, deployed from the same code and switched by the gateway. **Quota exhaustion**:

since May 2026 Microsoft has been moving quota to the subscription level, where Global Standard deployments of one model version share one pool across all regions and Data Zone Standard shares one pool per data zone; models not yet moved keep regional pools. A fallback in another region of the same subscription may drain the same pool, so real independence means another deployment type, reserved PTU or another subscription, and the register records which. **Reserved capacity**: PTU does not follow you to another region, so the contract says what the secondary runs on. **The gateway itself**: at the time of writing, multi-region API Management needs the classic Premium tier, while the v2 tiers offer availability zones in one region; choose with the recovery target in view. Agent state, indexes and conversation stores need an answer too: back up, or rebuild from source. Test failover and failback each quarter, in test first.

Semantic caching, carefully

The `llm-semantic-cache-lookup` and `llm-semantic-cache-store` policies return a stored answer when a new prompt is close enough to an earlier one, using an embeddings deployment and an external cache such as Azure Managed Redis. It can cut tokens and latency for repeated questions. Microsoft is explicit that because matching is by similarity, a cache can return responses that are incorrect, outdated or unsafe for the current request. Our rule: cache only for T1 use cases over shared, non-personal content, always partition the cache with `vary-by` so one user never receives another's answer, and keep the score threshold low; in this policy a lower value demands closer similarity.

The gateway inside Foundry

Microsoft has started to bring the gateway into the Foundry portal: an API Management instance can be associated with a Foundry resource to set token limits for deployments, register agents running elsewhere, and govern MCP tools. This is in preview at the time of writing. It changes where you click, not the pattern. The contract is the same: one door, a budget per team, a metric per call, and a pool behind it.

> **Every number adds up** *when every token passes the same door, is counted against a named budget, and lands in a report a finance lead can read without asking an engineer.*

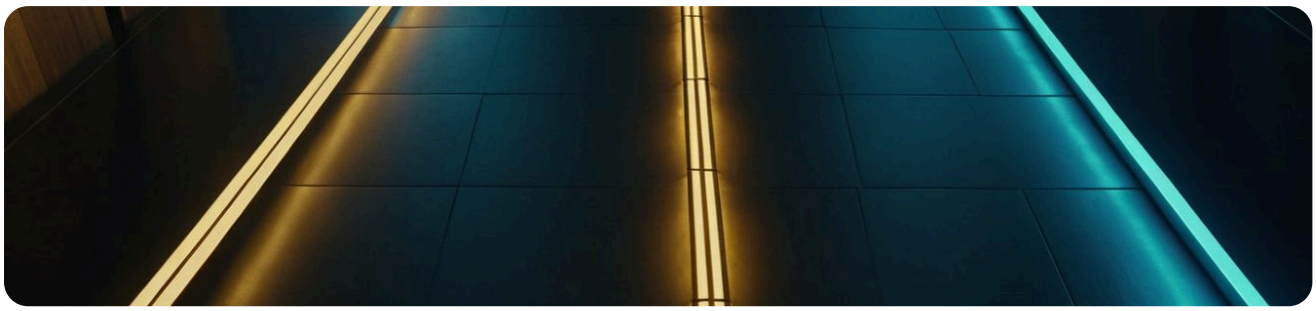
CONTRACT CHECKLIST

- All production application traffic passes the AI gateway: the Foundry private endpoint admits only the gateway subnet, and the only other callers are the listed exceptions, each with its own TPM cap.
- Callers authenticate to the gateway with Entra tokens validated by policy; the gateway authenticates to Foundry with managed identity.
- Every team has a monthly token quota and every application a per-minute rate, keyed by product and subscription.
- Token metrics by team and application feed the monthly showback report.
- Every fallback has a named quota failure domain, pools list only allowed regions, and failover is tested each quarter.
- Semantic caching is used only where the safety tier allows it, partitioned per user or group.

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

CHAPTER TEN

Agents by Contract



A chat assistant answers. An agent acts. It reads a case, calls a tool, writes a record, sends a message, and sometimes asks another agent to do part of the job. Each of those is a door, and each door needs the same answers this series has asked from the start: who may cross, what may cross, what must be true afterward, and who finds out. This chapter puts agents on Microsoft Foundry under contract: an identity per agent, a short list of tools, a person in the loop for actions that cannot be undone, and a fixed path from pilot to production.

Everything in this book comes together here. The landing zone (chapter 2) decides where the agent runs. Roles (chapter 3) decide who can build and publish it. The approved-model register and quota (chapter 4) decide which model it uses. Private networking (chapter 5) decides what it can reach. Guardrails (chapter 6), evaluations (chapter 7), tracing (chapter 8) and the gateway (chapter 9) decide what it may say, how well it must perform, what is recorded and what it may spend. The agent contract only adds what is specific to acting.

What Foundry Agent Service provides

Foundry Agent Service is the managed platform for building and running agents in Foundry. At the time of writing it offers **prompt agents**, defined by configuration (instructions, model, tools) and run by Foundry; **hosted agents**, your own code in a container that Foundry runs with a managed endpoint, scaling and identity; and voice-based prompt agents, which are in preview. Agents act through **tools**: built-in ones such as web search, file search, code interpreter and memory, and custom ones through functions, OpenAPI specifications and MCP servers. A **toolbox** groups a curated set of tools behind one managed endpoint with central authentication and versioning. Agents are versioned automatically and can be published to a stable endpoint.

For governance the important part is not the list of features. It is that each of them gives you a place to enforce a clause. The table maps the agent contract to where Foundry enforces it.

Clause	The question	Enforced by
Identity	Who is this agent, and who answers for it?	Microsoft Entra agent identity, with owner and sponsor
Scope	Which tools, which data, which actions?	Tool configuration and allowed tool list; RBAC on the agent identity
Obligations	What must be true before an action runs?	Human approval on the tool; evaluation gate before release
Prohibitions	What must never pass?	Agent guardrail on input, tool call, tool response and output
Evidence	What proves what happened?	Traces, Entra sign-in and audit logs, approval records
Budget	How much may it spend?	Gateway token limits; run and tool-call limits in code

One identity per agent

Foundry uses **Microsoft Entra agent identities**, a special kind of service principal designed for AI agents, created under an *agent identity blueprint*. In the current agent model, creating an agent provisions its own blueprint and the identity for its default instance. The blueprint is the governing template and parent object, and it can parent more than one identity, so the contract names the identity each agent instance runs as, not only the blueprint. Older agents can still share a project-level identity; Microsoft documents a migration path, and existing role assignments do not transfer, so plan the new grants before you migrate.

The most useful fact for a governance team is this: a newly created agent identity starts with no resource permissions, and Foundry does not assign roles to it automatically. That is least privilege by default, and it is the right starting point. The agent owner requests exactly the roles each tool needs, at the lowest scope, through the same request process as any workload identity in *Governing Azure by Contract*, chapter 5. Agent identities also appear in the tenant-wide inventory in the Microsoft Entra admin center, where, at the time of writing, Conditional Access, ID Protection, and owners and sponsors with expiration can be applied. Acting on behalf of a signed-in user through the agent identity is in preview at the time of writing; until it is generally available, we allow it only in pilots.

> **An empty allow list is not empty.** For MCP tools, Microsoft documents that setting `allowed_tools` to an empty list allows every tool the server exposes, not zero tools. Always list the tools by name, and treat tool descriptions and tool results from remote servers as untrusted input that can carry injected instructions.

A person in the loop for what cannot be undone

MCP tools in Foundry can be configured with `require_approval` set to `always`. The agent then stops and returns an approval request naming the tool and its arguments, and continues only when the application sends an approval response. Microsoft's guidance is to require approval for tools that write data or change resources, to review the tool name and arguments before approving, and to log approvals. We turn that into a rule: any tool that writes, sends, pays or deletes requires approval in production unless the contract names a tolerance and a reversal path that make approval unnecessary. The approval screen is part of the product, owned by the business, not a developer's console.

The agent contract fits on one page. The format is ours; the settings it refers to are Foundry's.

```
contract: claims-reply-agent            version: 4   tier: T3
identity:   own Entra agent identity; owner: claims ops lead; sponsor: head of claims
model:      deployment chat-prod (approved-model register entry 12)
tools:
  claims-mcp:  allowed_tools [get_claim, draft_reply, send_reply]
               require_approval: always for send_reply
  file_search: policy library index (read only)
rbac:       reader on claims index; no write roles outside the claims MCP server
guardrail:  claims-agent-guardrail v3 (input, tool call, tool response, output)
evaluation: gate eval-gate.yaml v7 must pass before publish
budget:     team quota via gateway; max 15 tool calls per run
evidence:   traces to appi-claims-prod; approvals logged with approver and arguments
delegation: may call agent policy-lookup-agent v2 only; callee runs under its own
contract
escalation: on guardrail block or failed tool, hand case to a handler queue
```

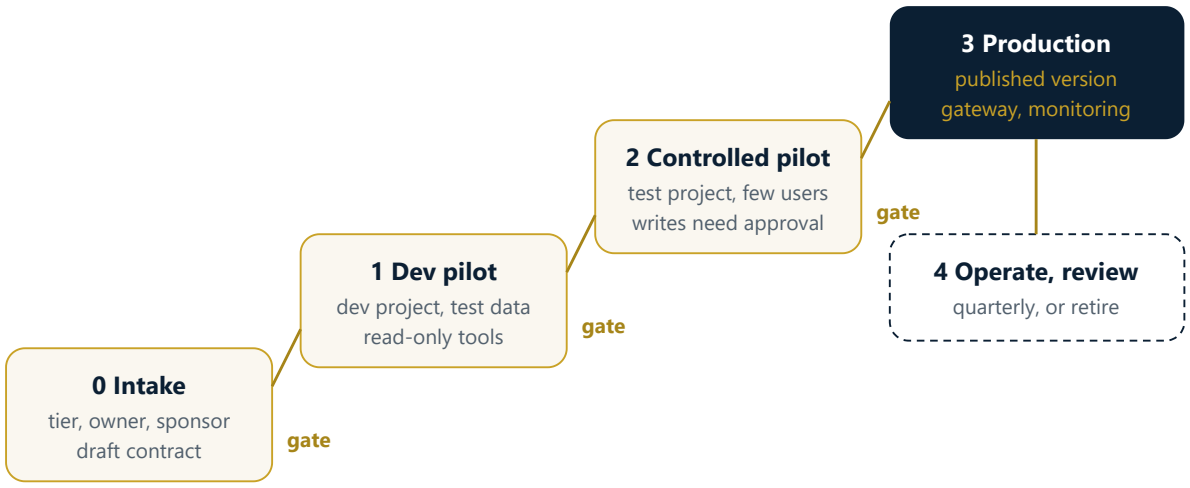
When one agent calls another, each keeps its own identity, tool list and guardrail, and the caller's contract names every agent it may call. The callee's spend and approvals are governed by the callee's contract, not inherited, so a delegation chain is never wider than the narrowest contract in it. Tools are software supply chain too: pin MCP servers, OpenAPI specifications and hosted-agent images to reviewed versions, scan images before they are published, and treat a new tool version as a change that passes the evaluation gate.

From pilot to production

Most agent problems in enterprise estates are not model problems. They are promotion problems: a pilot built with broad permissions quietly becomes the production system. A fixed path prevents that. Each stage has an entry gate, and nothing moves forward because a demo went well.

FIGURE 10.1

The agent promotion path



Each step up needs a gate. Permissions widen only at a gate, never between them. Stage 4 is not the end: the quarterly review can narrow scope or retire the agent.

1. **Intake.** The use case declares its safety tier, business owner, sponsor and outcome. A draft contract exists before any code.
2. **Dev pilot.** In the dev Foundry project, with synthetic or approved test data, read-only tools only. Gate to the next stage: golden set written, first evaluation run, tool list agreed.
3. **Controlled pilot.** In the test project, with a small named user group. Write tools allowed only with approval. Gate: evaluation thresholds met, red team findings closed, RBAC reviewed by security, contract signed by the owner.
4. **Production.** A published agent version, deployed by the pipeline into the production project, behind the gateway, with tracing, continuous evaluation and alerts on. No portal edits in production.
5. **Operate and review.** Each quarter: access review of the agent identity, evaluation trend, cost against value. The outcome is keep, narrow, or retire. Retiring removes the identity and its role assignments, not only the endpoint.

THE RECONCILIATION AGENT

A finance team piloted an agent that matched invoices to payments. In the dev pilot it only read data. In the controlled pilot it could propose ledger adjustments, each approved by an analyst, and every approval was traced. After a quarter the team saw that small adjustments were approved almost every time and large ones were often changed. The contract version 3 removed approval for adjustments below a set amount with a daily reversal report, and kept approval above it. The rule changed because the evidence supported it, not because someone was tired of clicking.

When an agent must stop

Every production agent needs a stop procedure that has been run at least once in test. It has four steps, each owned by a named role: block the agent's route or the team's product at the gateway, disable or unpublish the agent version, revoke the agent identity's role assignments or disable the identity in Entra, and preserve traces, approval records and sign-in logs before retention runs. Security operations can start it from a Defender alert (chapter 8); the business owner decides when the agent returns, and the return goes through the gate again.

The stop procedure sits inside a short AI incident runbook. It sets a **severity** scale tied to the safety tier, so a T3 agent writing wrong data is a high-severity incident by default. It names the **suspension criteria** agreed in advance, such as a guardrail bypass, an unapproved write or a cost spike past the team ceiling, so nobody debates them during the incident. It makes **evidence preservation** the first step after containment. It routes any exposure of personal or regulated data to the **privacy and legal** owners within the time your notification duties require. And it states the **rollback** target: the last agent, prompt and guardrail versions that passed the gate.

> **The model proposes. The contract decides.** *For an agent, that is literal: the agent proposes an action, and the identity, the tool list, the guardrail and the approval step decide whether it happens.*

CONTRACT CHECKLIST

- Every agent has its own Entra agent identity with a named owner and sponsor, and only the roles its tools need.
- Every tool connection lists allowed tools by name; no empty lists.
- Tools that write, send, pay or delete require human approval in production, and approvals are logged.
- Every agent has an explicit agent guardrail covering tool calls and tool responses.
- Agents reach production only through the pipeline, after passing the evaluation gate in the test project.
- Each production agent has a quarterly review that ends in keep, narrow or retire.

See it live: hub.jcardena.com/azure/azure-ai-works.html

APPENDIX

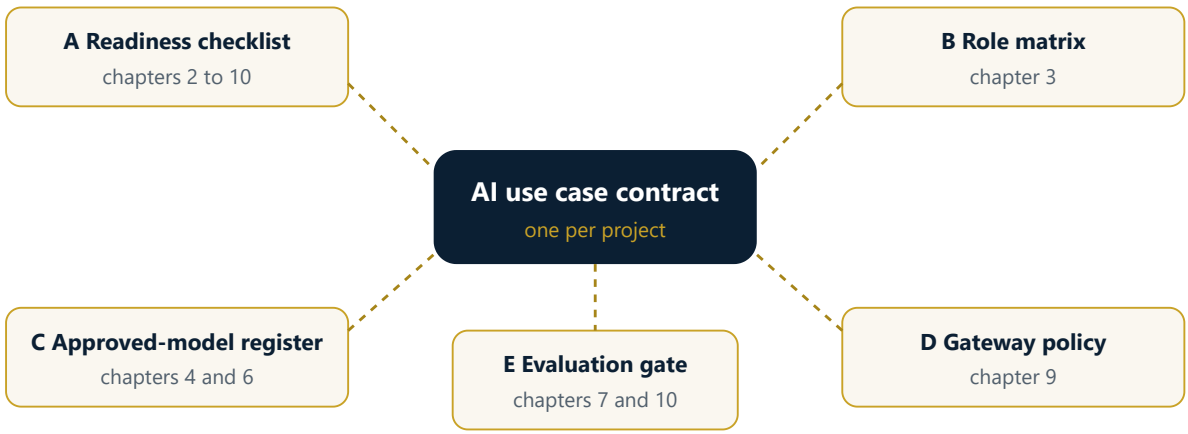
Appendix: Templates



These templates are the working parts of the book. Each one turns a chapter into something a team can fill in, review and keep under version control. Copy them into your repository, replace the examples with your own values, and treat every change to them as a change to the contract: a pull request, a named approver, and a record. Product names and preview features will move; the questions each template asks will not.

FIGURE 11.1

Five templates, one contract



Each template feeds the use case contract. A project is ready for production when all five are filled in for it and every field has an owner.

A. Foundry readiness checklist

Run this before the first project is created in a new Foundry environment, and again at each quarterly review. Every line is a yes or no, with a link to the evidence.

Area	Check	Evidence
Landing zone	Foundry resources live in landing zone subscriptions per environment, under the policies of *Governing Azure by Contract*, chapter 6	Management group and subscription map
Regions	Allowed regions for Foundry and gateway backends match the data residency rule	Allowed-locations policy assignment
Identity	Key-based access disabled where supported; apps and gateway use managed identity	Policy compliance report
Roles	Role matrix (B) applied by group; privileged roles eligible through PIM in production	Access review record
Network	Public network access disabled in dev, test and production (Sandbox excepted); private endpoints and DNS in place; in production the Foundry endpoint admits only the gateway subnet	Network diagram, policy compliance
Models	Every deployment maps to an approved-model register entry (C)	Register export vs deployment inventory
Safety	Every deployment and agent has an explicitly assigned guardrail	Guardrail assignment list
Visibility	Application Insights connected per project environment; alerts routed to named roles	Workbook, action groups
Cost	All production application-originated model traffic passes the AI gateway with team quotas (D), and the documented exceptions in chapter 9 have their own TPM caps; PTU utilization, Marketplace and platform costs are in the monthly report	Gateway metrics by team, Cost Management report
Encryption	Customer-managed keys where the data owner requires them, for Foundry and each dependency that stores AI data	CMK policy compliance
Threat detection	Defender AI posture and threat protection on for every Foundry subscription; alerts owned by security operations	Defender plan status, alert routing
Data governance	Retention, classification and deletion set for prompts, traces and LLM logs; document-level security trimming tested for every retrieval index	Retention settings, trimming test in the golden set
Continuity	Recovery time and point written per use case; second region and fallback quota in place where required; failover tested this quarter	Failover test record

B. Role matrix

The same matrix as chapter 3, using the built-in Foundry roles at the time of writing. Microsoft renamed them recently (Foundry User was Azure AI User, and so on), and recommends using role definition IDs rather than names in code during the rename. The role IDs are listed on the Microsoft Learn page for Foundry role-based access control. Assign to groups, never to individuals, as in *Governing Azure by Contract*, chapter 5.

Persona	Dev	Test	Prod
Platform team	Foundry Account Owner, standing	Foundry Account Owner via PIM	Foundry Account Owner via PIM, approval required
Use case lead	Foundry Project Manager	Foundry Project Manager via PIM	Reader; Foundry User via PIM for incidents
Builders	Foundry User on the project; Reader on the resource	Reader; Foundry User on the project via PIM for debugging	Reader only
Deployment pipeline (workload identity)	Foundry Account Owner to deploy models; Foundry Project Manager to publish agents	Same	Same, from the protected release pipeline only
Gateway or model-calling app	Foundry User on the Foundry resource, or a custom inference-only role	Same	Gateway identity only; apps call the gateway (chapter 9)
Agent-calling app	Foundry Agent Consumer at project or agent scope	Same	Same, agent scope preferred
Project identity	Foundry User on the Foundry resource (needed for continuous evaluation)	Same	Same
Responsible AI owner, auditor	Reader; Log Analytics Reader	Same	Same; content tables through PIM
FinOps lead	Cognitive Services Usages Reader at subscription scope, plus Cost Management Reader as in Governing Azure by Contract, chapter 10		

Agent identities are not in this table on purpose. Each agent's roles are listed in its own contract (chapter 10), because each agent needs a different, narrow set.

C. Approved-model register

One row per approved model version and deployment type. A deployment that does not match a row is a policy violation, not a convenience.

Field	Example
Entry ID	12
Model and version	Chat model, pinned version (no automatic upgrade in production)
Deployment type	Standard regional, or provisioned throughput
Base model and training data	Fine-tuned models only: base model entry, dataset version, owner and classification
Allowed regions	Primary and secondary, both inside the residency boundary
Allowed safety tiers	T1, T2
Required guardrail	Organization baseline guardrail v3 or stricter
Evaluation evidence	Link to the gate run that approved this version
Owner and reviewer	AI platform owner; responsible AI owner
Review and retirement dates	Next review date; provider retirement date when announced
Status	Approved, restricted, deprecated or blocked

D. Gateway policy sample

An API-level policy combining chapter 9. All numeric values are examples, not recommendations. Callers present an Entra token, validated first; the subscription is the metering key. Set every AI API to require an API Management subscription, so both token-limit keys always have a value. The managed identity audience `https://cognitiveservices.azure.com` is the one Microsoft documents for model inference endpoints; Foundry project and agent endpoints use a different audience, so check it if the gateway fronts agents. The content safety check here is an extra door for traffic that Foundry guardrails do not cover; it does not replace the guardrail on each deployment (chapter 6). The content safety backend points to an Azure AI Content Safety resource; the pool and backends are defined in infrastructure as code. With `FourSeverityLevels`, a threshold of 4 blocks medium and high severity.

```
<policies>
    <inbound>
        <base />
        <validate-azure-ad-token tenant-id="{{tenant-id}}">
            <client-application-ids>
                <application-id>{{claims-app-client-id}}</application-id>
            </client-application-ids>
        </validate-azure-ad-token>
        <authentication-managed-identity resource="https://cognitiveservices.azure.com"
/>
        <llm-content-safety backend-id="content-safety" shield-prompt="true" enforce-on-
completions="true">
            <categories output-type="FourSeverityLevels">
                <category name="Hate" threshold="4" />
                <category name="Sexual" threshold="4" />
                <category name="SelfHarm" threshold="4" />
                <category name="Violence" threshold="4" />
            </categories>
        </llm-content-safety>
        <!-- Example values only -->
        <llm-token-limit counter-key="@(context.Subscription.Id)"
            tokens-per-minute="20000" estimate-prompt-tokens="false" />
        <llm-token-limit counter-key="@(context.Product.Id)"
            token-quota="50000000" token-quota-period="Monthly"
            estimate-prompt-tokens="false"
            remaining-quota-tokens-header-name="x-team-tokens-remaining" />
        <llm-emit-token-metric namespace="ai-gateway">
            <dimension name="Product ID" />
            <dimension name="Subscription ID" />
            <dimension name="API ID" />
        </llm-emit-token-metric>
        <set-backend-service backend-id="foundry-chat-pool" />
    </inbound>
    <backend>
        <base />
    </backend>
    <outbound>
        <base />
    </outbound>
    <on-error>
        <base />
    </on-error>
</policies>
```

E. Evaluation gate checklist

Use this at every promotion from dev to test and from test to production, for models, prompts and agents alike.

1. The golden set version is named, owned by the business owner, and includes every confirmed production failure since the last release.
2. Thresholds are committed in the repository before the run, signed off by the use case owner.
3. Quality evaluators for the use case ran (for example groundedness and relevance for retrieval, tool call accuracy for agents).
4. Risk and safety evaluators ran, with zero tolerated defects for the harm categories and indirect attacks.
5. The judge deployment is separate from the system under test and was calibrated against human-graded samples.
6. For agents: red team findings are closed or accepted in writing by the responsible AI owner, and every write tool requires approval.
7. Results are stored in the Foundry project and linked from the release record and the approved-model register.
8. Continuous evaluation for the new version is configured before traffic is switched.

Check preview status before you depend on it. Several features these templates refer to were in preview at the time of writing, including agent guardrails on tool calls, several agent evaluators, recurring evaluations and the AI gateway inside Foundry. Record the status in the register and recheck it at each quarterly review.

CONTRACT CHECKLIST

- All five templates live in a repository with pull request review.
- Each production AI project links to its row in the register, its role assignments, its gateway product and its last gate run.
- The readiness checklist is rerun every quarter, and failed lines become tracked work items.
- Preview features in use are listed with an owner who tracks their status.

See it live: the Foundry governance page at hub.jcardena.com/azure/foundry-governance.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.

Book 2 **AI Foundry by Contract**: delivering governed AI on Azure. (This book.)

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/ai-foundry-by-contract. See it live at hub.jcardena.com/azure.

GDCD on Azure · Book 2 · Trust, earned at every layer.