



Foundry before bed

*A slow walk through Microsoft Foundry,
its agents and the Citadel around them*

BY DIEGO · SEPTEMBER 2026 · 20 MINUTES

BEFORE YOU START

How to read this

Twenty minutes. One cup of something warm.

No code to run. No slides.

When a technical word shows up, it gets explained right there. If you fall asleep in chapter six, fine. Chapter six will still be there tomorrow.

I wrote this for myself first. I kept losing track of what got renamed, what got merged and which box goes where on the whiteboard. So this is the version I wish somebody had handed me a year ago.

It's probably wrong in two places. This stuff moves every month. Check the docs before you quote me to a customer.

It's still ok.

The whiteboard problem

Or why nobody can explain Foundry in one sentence

Someone asks what Foundry is. You draw a box. Then another box inside it. Then an arrow to a third box labelled "stuff".

Five minutes later there are nine boxes and nobody in the room is sure which one sends the invoice.

Classic.

Here's the short version I use now.

Foundry is one Azure resource where your agents live, together with the models and tools they use.

One resource. One endpoint. One set of permissions. Everything else hangs off that.

Half the confusion is the names. Azure AI Studio became Azure AI Foundry. Azure AI Foundry became Microsoft Foundry. Azure AI Services became Foundry Tools. The old "hub" setup still exists and is now called Foundry classic. New things land in the new portal at ai.azure.com.

If a customer shows you a hub, they're on the old model. Not a crime. Just know which picture you're drawing before you pick up the marker.

The house and the rooms

Two words that matter more than the rest

Resource and project.

The **Foundry resource** is the house. It sits in an Azure subscription, in a region. Networking, keys, policies and billing attach to it.

A **project** is a room in that house. A team gets a project. Inside it they deploy models, build agents, keep files and run tests. Projects stop teams from tripping over each other while they share the same front door.



Each project has one endpoint. That's the address your code talks to.

A year or two ago there were five endpoints and four SDK packages to juggle. Now there's one client library (azure-ai-projects) and the familiar OpenAI client pointed at the project. Less wiring. Fewer ways to get it wrong.

Coming from Azure OpenAI? You can upgrade that resource in place into a Foundry resource. Same endpoint, same keys, same deployments. That's usually the easiest first conversation with a customer who already has GPT running somewhere and is nervous about "migrating to Foundry".

There's nothing to migrate. You flip it.

Models, the engine room

Ten thousand engines and you'll use three

A model is the thing that reads text and predicts more text. The Foundry catalog has more than 10,000 of them. OpenAI's GPT family, Microsoft's own, Anthropic's Claude, Meta, Mistral, DeepSeek and a long tail of open models nobody has heard of.

Nobody needs 10,000. Most customers end up with three or four.

A **token**, since it's about to come up a lot, is a chunk of a word. Roughly four characters of English. Everything is priced and throttled in tokens: input tokens, output tokens, tokens per minute.

A **deployment** is how a catalog entry becomes something you can call. You pick a model and a deployment type. The type decides where the work happens and how you pay.

Global Standard. Runs wherever Azure has capacity.
Cheapest and easiest to get.

Data Zone Standard. Processing stays inside a zone, like the EU. The one Belgian customers ask about first.

Standard. Regional. Pay per token.

Provisioned (PTU). Reserved capacity paid per hour, used or not. Predictable speed, predictable bill.

Batch. For jobs that can wait up to a day. About half the price.

(Stored data stays in the geography you picked in all of these. The type is about where the thinking happens.)

Two more things worth knowing.

Model router is a deployment that picks the model for you, request by request. Easy question, small cheap model. Hard reasoning, big model. You give up some control and get a lower bill. For most chat workloads that's a good deal.

Fine-tuning exists and customers love to ask about it. Most of them need better instructions and better retrieval first. Fine-tuning is for when you did both and there's still a gap.

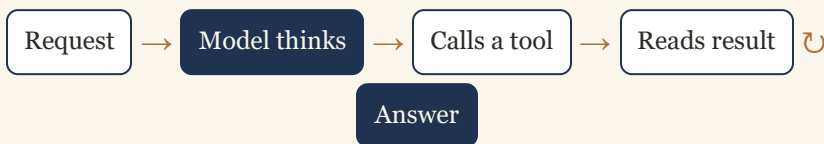
There's also Foundry Local, which runs small models on a laptop with no cloud at all. Nice for demos on bad hotel wifi. That's about all I'll say about it tonight.

So what is an agent anyway

A model with a job description and hands

The job description is the instructions. The hands are tools. Put them in a loop and you have an agent.

The loop goes like this.



Repeat the middle until the model decides it's done

A request comes in. The model reads it together with its instructions. It decides it needs something, say a customer's order history. It calls a tool. The tool answers. The model reads the answer and decides again. Maybe another tool. Maybe it's done. Then it replies.

That's it. Everything else in this booklet is plumbing around that loop.

Foundry Agent Service is the managed runtime that runs the loop for you. It keeps the conversation, calls the tools, handles identity and logs what happened.

It's built on the Responses API (the newer OpenAI-style API where one call can include reasoning and tool use). If you

remember the older Assistants API with threads, messages and runs: those became conversations, items and responses. Same ideas, new names. Naturally.

There are three flavours.

Prompt agents. Declarative. Pick a model, write instructions, tick some tools. Foundry hosts it. No code, no container. Ten minutes in the portal. There's a voice version too for spoken conversations.

Hosted agents. Your own code in any framework, packaged as a container. Foundry runs it with an endpoint, scaling, identity and tracing. For when the logic is too specific to fit in a prompt.

Workflows. Several agents and steps wired together in a known order, with branches and a human approval in the middle if you want. For processes that already have a shape.

My rule of thumb: start with a prompt agent. Move to hosted when the prompt agent visibly can't do the job.

Most people skip straight to code because code feels serious. Then they spend three weeks building what the portal did in an afternoon. I've done it. Twice.

Tools, or giving the thing hands

A model on its own can only talk

Tools let it do things.

Function calling is the base mechanism. You describe a function: its name, what it does, what inputs it takes. The model decides when to call it and fills in the inputs. Your code runs it. The model never runs anything itself. It asks politely and waits.

Foundry ships a set of built-in tools. Code interpreter runs Python in a sandbox, so the model does real maths instead of guessing it. File search reads uploaded documents. Bing grounding checks the web. There are connectors for SharePoint and Fabric data agents. OpenAPI tools turn any REST API with a spec into something the agent can call.

Then there's MCP.

MCP (Model Context Protocol) is a standard way for an agent to discover and call tools. Think USB-C for tools. Build a tool server once and any agent that speaks MCP can plug into it. Anthropic started it. Pretty much everybody adopted it within a year, which in this industry counts as a miracle.

MCP solves the plug. It doesn't solve the mess.

Take a company with forty agents. Each one wires its own tools. Each one keeps its own secrets and refreshes its own tokens.

Nobody can tell you which agent is allowed to send emails.

Foundry Toolbox is the fix for that. You define a curated set of tools once and put them behind a single MCP endpoint.

Credentials live in one place. Agents use the toolbox instead of wiring tools one by one. Change how a tool works underneath and no agent code has to change.

Toolbox also does tool search. An agent with 200 tools available only loads the handful it needs for this request. Less confusion for the model, fewer tokens for you.

And then **A2A** (Agent2Agent), the other protocol you'll hear. MCP is agent to tool. A2A is agent to agent. One agent hands a task to another agent and gets the result back. The other one may be built by another team on another platform.

Useful. Also a great way to build a system nobody can debug. Use it sparingly.

Knowledge, or how it stops making things up

Open book instead of memory test

Models know what they were trained on. They don't know your contracts, your price list or what happened last Tuesday.

RAG (retrieval-augmented generation) fixes that. Before the model answers, you search your own documents and paste the relevant bits into the prompt. The model answers from those bits. An open-book exam.

In Azure the search part is usually **Azure AI Search**. It cuts documents into chunks, turns each chunk into a vector (a list of numbers that captures meaning, so "car" sits close to "vehicle") and finds the chunks closest to the question.

Classic RAG does one search and hopes.

Agentic retrieval does better. The model splits a messy question into smaller ones, searches for each, checks what came back and merges it. Slower. Much better on questions like "compare our 2024 and 2025 terms for Belgian customers".

Foundry IQ is the packaged version. You create a knowledge base, point it at sources (search indexes, SharePoint, blob storage, the web) and any agent can use it as one tool. Build the knowledge

once and share it across agents. Same idea as Toolbox, applied to data.

Memory is the other half. Knowledge is what the company knows. Memory is what the agent remembers about this particular user across conversations. "Prefers answers in Dutch." "Works on the Antwerp site." Foundry has managed memory in preview. Handy. Also the first thing a data protection officer will ask about, so know where it's stored.

Most agent failures I've seen are retrieval failures in a model costume.

The model was fine. It just got the wrong three paragraphs.

Frameworks and the multi-agent temptation

Five agents in a trench coat

At some point you'll write code. You'll want a framework.

Microsoft Agent Framework is the one Microsoft is putting its weight behind. It's the successor to both Semantic Kernel and AutoGen, which used to be two separate projects with overlapping ideas and two sets of fans. Now it's one open-source SDK for Python and .NET, with agents, tools, workflows and multi-agent patterns in the box.

Foundry also runs LangGraph, CrewAI, the OpenAI Agents SDK or plain code. A hosted agent doesn't care what's inside the container.

Now the temptation.

Once you have one agent you want five. A planner, a researcher, a writer, a critic and a manager to keep them in line. It looks great on a slide. Very org chart.

Each hop adds time, cost and one more place for things to go wrong. Five agents passing messages is five chances to misunderstand each other.

Just like people.

Use one agent with good tools until it clearly breaks. Split when there's a real reason: a different security boundary, a different owner, a genuinely different skill. "It looks cleaner" doesn't count.

Does it work? Nobody knows until you measure

The uncomfortable chapter

You can't unit test an agent the way you test normal code. Same input, different output. That's the design.

So you evaluate.

An **evaluation** runs a set of test questions through the agent and scores the answers. Some scores are simple. Did it call the right tool? Did it finish the task? Some use another model as a judge. Is the answer grounded in the text it retrieved? Is it relevant? Foundry ships built-in evaluators for these, plus safety ones for harmful content and leaking protected material.

Build a test set early. Fifty real questions from real users beat five hundred synthetic ones.

Tracing is the second habit. Every run gets recorded step by step: what the model saw, what it decided, which tool it called, what came back, how long it took, how many tokens it burned. Foundry uses OpenTelemetry (the open standard for this kind of logging) and sends it to Application Insights.

When a user says "it gave me nonsense on Thursday", tracing is how you find out why in five minutes instead of an afternoon.

Red teaming is the third. The AI Red Teaming Agent attacks your agent on purpose: jailbreaks, prompt injection (hiding instructions inside a document so the agent obeys the document instead of you), attempts to make it say or do what it shouldn't. Better you find it than a journalist.

Then keep going in production. Sample live traffic, score it, alert when quality drops. Models get updated. Data changes. An agent that was fine in June can be quietly worse by September and nobody notices until a customer does.

CHAPTER 9

The Citadel, or why security wants a word

From one agent to two hundred

Everything so far is about building one agent well.

The Citadel is about what happens when you have two hundred. Built by twelve teams. Calling six model providers. Spending money nobody can trace back to a cost centre.

AI Citadel, officially the Foundry Citadel Platform, is Microsoft's reference architecture for governing AI in a large company. It's a blueprint with open-source accelerators on GitHub (aka.ms/foundry-citadel). There's no SKU. You deploy the patterns and adapt them.

It stacks four layers.

Layer 4 Security Foundation · Defender, Purview, Entra

Layer 3 Agent Management · Entra Agent ID, Agent 365

Layer 2 Agent Operations · Foundry and its control plane

Layer 1 Governance Hub · the AI gateway on API Management

Each layer owns one job and can be upgraded on its own

The official motto is "separation of concerns with unified oversight". In plain words: every layer does one job. You can swap one out without breaking the others.

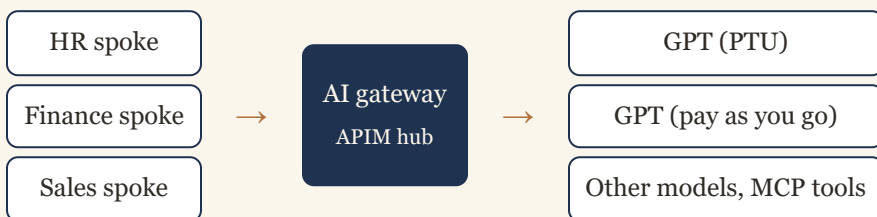
Next two chapters walk up the stack.

Layer 1, the gate in the wall

What people usually mean when they say "Citadel" in a meeting

The Governance Hub is **Azure API Management** (APIM) set up as an AI gateway. APIM is Azure's front door for APIs. Every call passes through it and it applies rules on the way.

The layout is hub and spoke. The hub is central and owned by the platform team. The spokes are the business units, each with their own Foundry projects and agents. Spokes build what they like. Every model call goes out through the hub.



Spokes build freely, everything leaves through one gate

What the gateway does, roughly in the order you'll care about it:

Identity. Checks who's calling with Entra tokens, instead of API keys scattered through code and old Teams chats.

Token limits. Each team or app gets a tokens-per-minute budget. One runaway agent can't eat everybody's quota.

Load balancing. Spreads traffic across deployments and regions. Fills the PTU you already pay for first, spills over to pay

as you go when it's full. This alone often pays for the gateway.

Semantic cache. If somebody asks nearly the same question as five minutes ago, answer from cache. Cheaper and faster.

Content safety. Screens prompts and answers centrally, instead of trusting every app to remember.

Cost attribution. Every token logged against a team, a product, a cost centre. Usage flows through Event Hub into Cosmos DB for reporting. Finally someone can answer "who spent 40,000 euros on GPT last month".

Tools can go through the gateway too, as MCP. Then the same rules apply to what agents do, on top of what they say.

You deploy it with Bicep or Terraform into an existing Azure landing zone, either inside the network hub or as its own spoke. Large regulated companies tend to pick the separate spoke.

imho: if a customer has more than a couple of teams using models, they need something in this position. It doesn't have to be the accelerator line by line. It does have to exist.

Layers 2 to 4, the quiet ones

Boring on purpose

Layer 2 is Foundry itself plus the **Foundry Control Plane**. The control plane is the fleet view: all your agents in one list, including ones built outside Foundry, with health, cost, evaluation scores and policy compliance.

Air traffic control. You don't fly the planes. You know where they are.

Layer 3 is identity for agents. This one took me a while.

Agents need their own identity. Running an agent under the developer's account is how you end up with an agent that can read the CEO's mailbox, because the developer once could.

Microsoft Entra Agent ID gives each agent its own identity in the directory. Like a user account, built for software that acts on its own. It gets permissions, an owner (called a sponsor), access reviews and an end date. When the project dies, the agent's access dies with it.

Agent 365 sits on top as the management layer. A registry of every agent in the company, including shadow agents: the ones somebody built on a Friday afternoon and never told IT about. It's where IT goes to find them, govern them or switch them off.

Layer 4 is the security products you already know, taught about AI.

Defender for Cloud spots attacks on AI workloads, like jailbreak and prompt injection attempts. It sends them to the SOC.

Purview tracks sensitive data going into and out of prompts, applies labels and data loss rules. It keeps the audit trail compliance wants.

Entra handles sign-in and conditional access for people and agents alike.

None of this is exciting. Doesn't make good click bait.

Boring security is security that works.

One question, start to finish

Everything so far in a single request

An employee in Ghent asks the HR agent: "How many holiday days do I have left? Can I take the week after Easter?"

- 1.** The request reaches the HR agent, a prompt agent in the HR team's Foundry project. That's a spoke.
- 2.** The agent signs in as itself, with its own Entra Agent ID.
- 3.** The model call leaves the spoke through the APIM gateway. Token budget checked. Content safety checked. Routed to an EU Data Zone deployment.
- 4.** The model decides it needs two things: the leave balance and the team calendar.
- 5.** It calls the HR toolbox over MCP. The toolbox holds the credentials for the HR system. The agent never sees a password.
- 6.** It checks the leave policy in a Foundry IQ knowledge base built on the HR handbook.
- 7.** It answers, with the policy paragraph cited.
- 8.** Every step lands in tracing. The tokens get charged to HR. Later, continuous evaluation samples the conversation and scores it.

9. Purview notes that personal data passed through. Defender watched for anything odd. Agent 365 knows the agent exists and who sponsors it.

Nine steps for one sentence of reply.

The employee sees none of it. Good.

Things I watched people trip on

Including me

Quota. Deployments have tokens-per-minute limits per region and subscription. Everyone finds out during the demo. Ask for more early, or use Global deployments and a gateway to spread load.

Networking. Private endpoints across Foundry, Search, Storage and Key Vault are doable and fiddly. Budget days. Most "Foundry doesn't work" tickets I've heard of turned out to be DNS.

Classic versus new. Hub projects, old SDKs, old API versions. Tutorials online mix them freely. Check the date on anything you copy.

Preview features. Plenty of what's in this booklet is preview. Fine for a pilot. Check the GA list before it goes into a production contract.

Cost surprises. Agents loop. A loop that calls a big reasoning model eight times per question costs eight times what the slide said. Trace it, measure it, use model router or smaller models where you can.

Skipping evaluation. Everyone plans to add it later. Later shows up as a production incident.

Governance too late. Retrofitting a gateway after twelve teams hard-coded endpoints and keys eats a quarter of somebody's life. Put it in while there are only two teams.

The pocket glossary

For when you wake up and forgot half of it

Foundry resource. The Azure resource that holds everything. The house.

Project. A team's space inside it, with one endpoint. A room.

Token. About four characters. The unit of cost and throttling.

Deployment. A model from the catalog made callable, with a type that sets location and pricing.

PTU. Provisioned throughput unit. Reserved model capacity paid per hour.

Model router. A deployment that picks the model per request.

Agent. Model plus instructions plus tools, running in a loop.

Prompt agent. Declarative agent hosted by Foundry. No code.

Hosted agent. Your own code in a container, run by Foundry.

Workflow. Several agents and steps in a known order.

MCP. Standard protocol for agents to find and call tools.

A2A. Protocol for agents to hand work to other agents.

Toolbox. Curated tools behind one managed MCP endpoint.

RAG. Search your documents first, then answer from what you found.

Foundry IQ. Shared knowledge bases for agents, built on Azure AI Search.

Microsoft Agent Framework. Open-source SDK, successor of Semantic Kernel and AutoGen.

Evaluation. Scoring agent answers against a test set.

Tracing. Step-by-step record of every agent run.

Citadel. Microsoft's four-layer reference architecture for governing AI.

AI gateway. API Management applying AI rules to every model and tool call.

Control plane. Fleet view of agents, cost, health and compliance.

Entra Agent ID. A directory identity for an agent, with a sponsor.

Agent 365. Registry and off switch for every agent in the company.

LIGHTS OUT

Before you close it

Foundry is where you build one agent.

The Citadel is how you live with two hundred.

Go to sleep.

TL;DR

Foundry is one Azure resource with projects inside, one endpoint per project and more than 10,000 models behind it.

An agent is a model with instructions and tools in a loop. Start with a prompt agent. Go to hosted code only when you have to.

MCP is the plug for tools. Toolbox keeps the tools and their secrets in one place. Foundry IQ does the same for knowledge.

Evaluate early, trace everything, red team before someone else does.

The Citadel adds four layers for scale: an APIM gateway, the Foundry control plane, agent identities through Entra and Agent 365. Defender plus Purview underneath.

Put the gateway in while there are two teams. Not twelve.

Next time: building the HR agent from chapter twelve for real, with the bill attached.

Sources: Microsoft Learn, "What is Microsoft Foundry?" and "What is Toolbox in Microsoft Foundry?" (both updated 2026). [Azure-Samples/foundry-citadel-platform](#) and [Azure-Samples/ai-hub-gateway-solution-accelerator](#) on GitHub. Written in personal capacity. Features and names change often, check before quoting.