MH

Independent enterprise handbook · Edition 1

Microsoft Foundry Adoption Guide for Enterprises

From first use case to governed AI operations

DECIDE | DESIGN | BUILD | ADOPT

Author
Marcel Haas

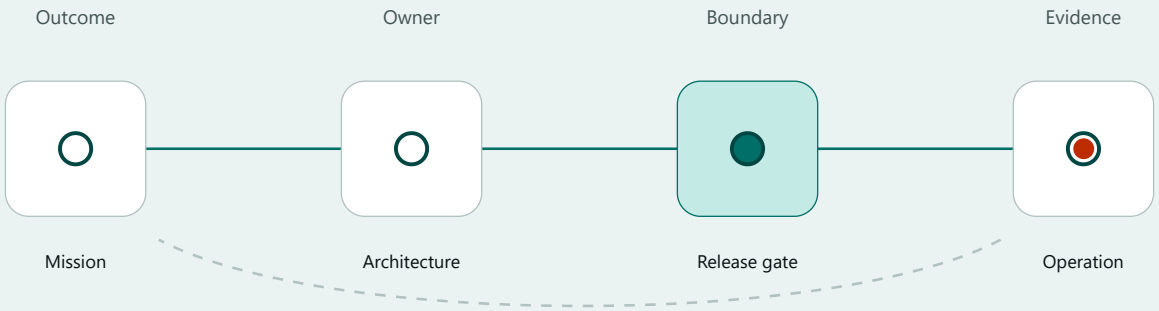
Publisher
MH – Applied AI

Published
28 July 2026

Version
1.0

Microsoft Foundry adoption succeeds when an enterprise starts with a bounded business mission, establishes identity, network, data, evaluation, and operating ownership before scale, and treats models and agents as replaceable components inside a governed workload.

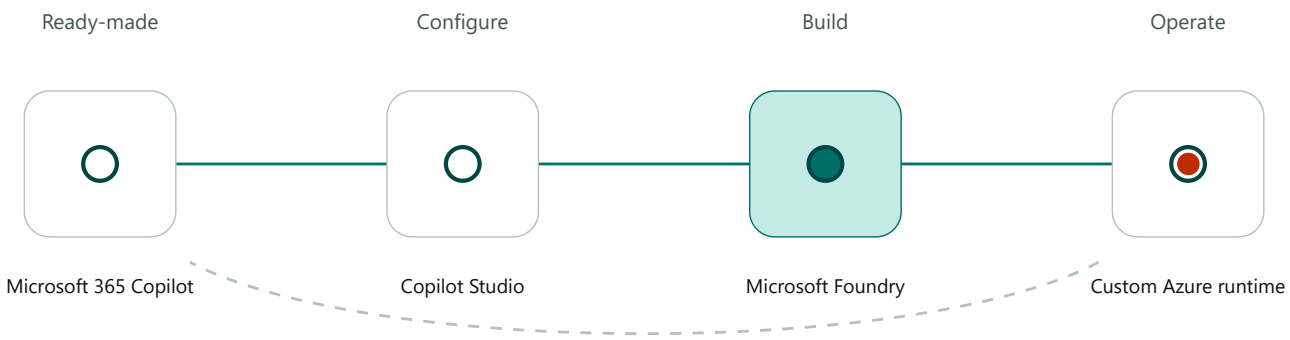
This independent guide is not sponsored or endorsed by Microsoft. Product facts are grounded in current Microsoft documentation; operating recommendations are MH – Applied AI interpretations.



A mission enters; a governed workload leaves

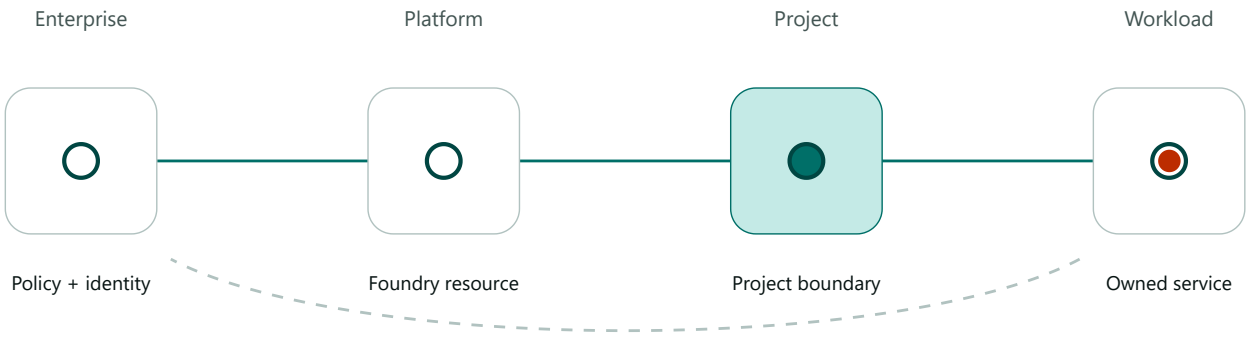
Business outcome, ownership, baseline, boundaries, evidence, and operation form a sequence. A model is selected only after the mission is bounded.

Contents



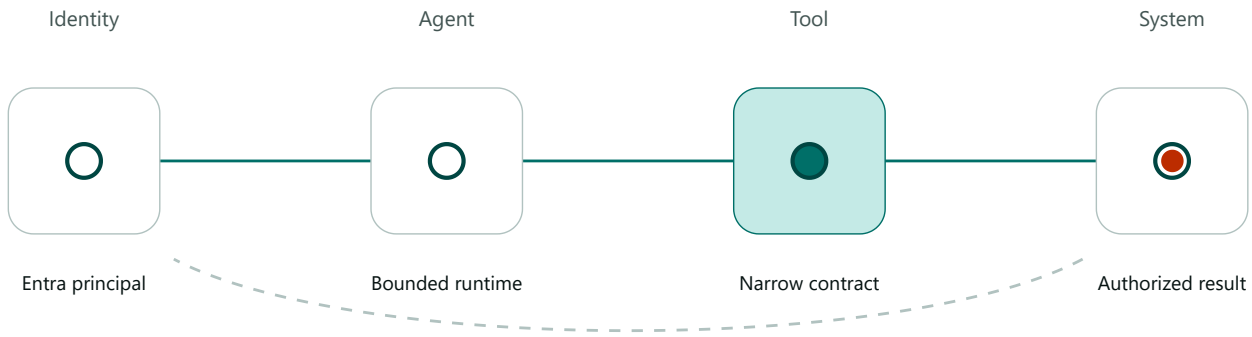
Choose the operating lane before the platform

A decision field moving from ready-made productivity tools through configurable agents and custom Foundry workloads to infrastructure-level control.



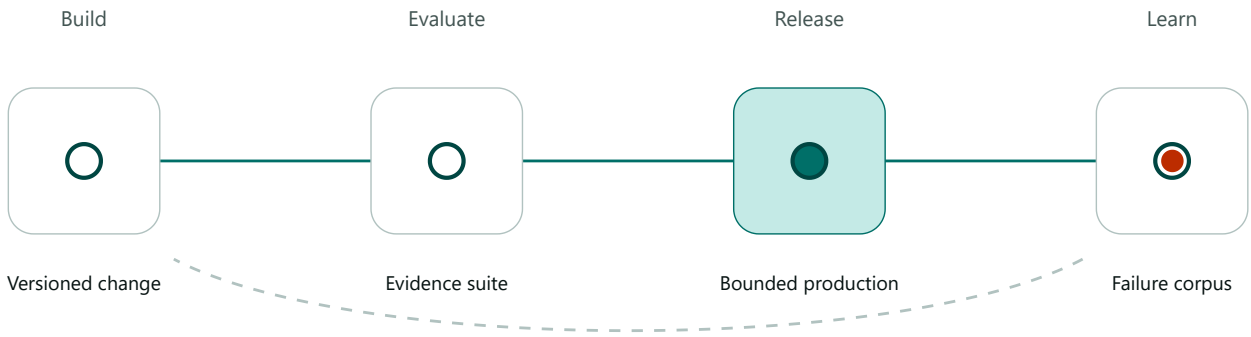
Foundry responsibility boundary

Enterprise policy surrounds platform and workload responsibilities. The Foundry resource contains projects; each project contains governed models, agents, tools, and evaluation assets.



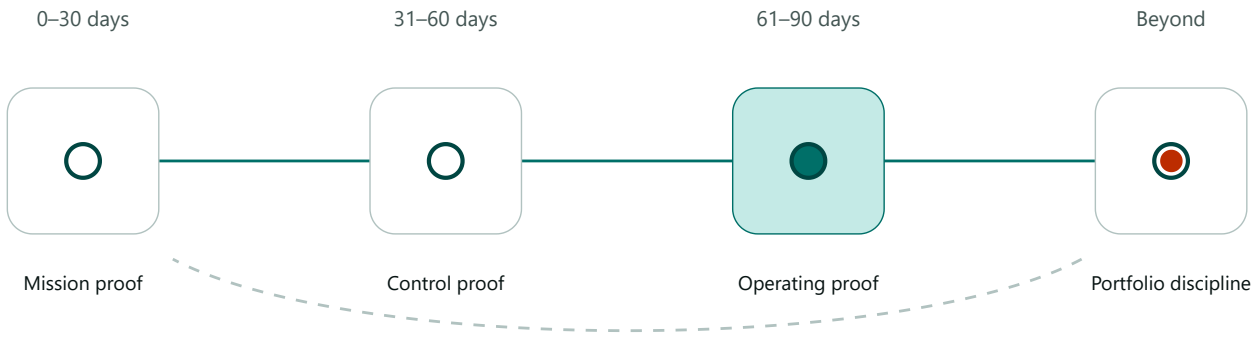
Identity follows every action

A user or workload identity reaches the agent, which can call only approved tools. Data and action systems independently validate authorization and return observable results.



Evaluation is the deployment gate

Representative cases move through build, evaluation, release, production observation, and failure learning. Every material change returns through the gate.



Ninety days, three proofs

The first month proves the mission, the second proves the controls, and the third proves operation. Scaling begins only after all three are visible.

Independent guide disclosure

This handbook is written and published by MH – Applied AI, the forward-deployed AI engineering practice founded by Marcel Haas. It is an independent field guide, not a Microsoft publication. Microsoft has not reviewed, sponsored, certified, or endorsed this document, and nothing here should be read as an official statement of Microsoft policy, roadmap, pricing, or support. Product names, capabilities, and URLs referenced below belong to Microsoft and change over time; verify every operational detail against current Microsoft Learn documentation before you design or budget a program around it. Where we state an operating recommendation rather than a documented product fact, we label it explicitly as an MH – Applied AI recommendation.

The guide exists because most enterprises do not fail at Microsoft Foundry because the platform is weak. They fail because they skip the governance and mission discipline that has to exist around any capable AI platform, and Foundry is capable enough to expose that gap quickly and expensively.

Executive brief

Microsoft Foundry is Microsoft's unified surface for building, evaluating, deploying, and operating generative AI applications and agents on Azure. It brings together model access, agent runtimes, tool and knowledge integration, evaluation, tracing, and observability under a more consistent management plane than the earlier generation of separate Azure AI services and the Azure AI Studio experience that preceded it. That consolidation is real and useful. It is also frequently mistaken for a strategy, and a platform cannot substitute for one.

Enterprises that adopt Foundry well share a small number of habits. They pick a single, bounded business mission before they touch the platform. They decide who owns identity, network isolation, data boundaries, model selection, evaluation, and production operations before the first pilot leaves a sandbox. They separate the parts of the system that are genuinely reusable (landing zone, identity model, evaluation harness, observability pipeline) from the parts that are disposable (a specific model, a specific prompt, a specific agent design), and they budget effort accordingly. They treat evaluation as a release gate with named owners and explicit thresholds, not as a demo step. They plan cost and capacity as an ongoing operating discipline, not a one-time estimate.

Enterprises that struggle usually did the opposite. They opened a Foundry project because a platform initiative needed a visible artifact, connected it to a broad set of internal data "to see what is possible," and discovered months later that no one owned the evaluation set, the network boundary was porous, the cost curve was unpredictable, and the pilot could not be safely retired or promoted because no one had assigned it an owner.

This handbook is organized as a working sequence: understand the landscape and its renamed history, choose an adoption lane that matches your organization's actual maturity, define one mission properly, stand up the operating model and landing zone that will outlast that first mission, secure identity and data, choose models and agents deliberately, gate everything through evaluation, operate with real observability, manage cost like an engineering discipline, and use a 90-day plan, a scorecard, and a decision pack to make the case internally. It closes with a glossary and a transparent methodology section, because a guide about evidence-based AI adoption should hold itself to the same standard.

Read this as a field manual, not a sales document. Wherever Foundry is the wrong tool for your constraints, the guide says so.

1. The Foundry landscape, named correctly

Names in this space have moved fast, and the vocabulary in your organization is probably a mix of eras. It is worth being precise, because search results, older architecture diagrams, and vendor conversations will keep surfacing the earlier terms for years.

Microsoft's current umbrella name is Microsoft Foundry. Before that, the same conceptual space was marketed as Azure AI Foundry, and before that as Azure AI Studio, with adjacent services such as Azure OpenAI Service, Azure AI Search, and Azure Machine Learning contributing overlapping but separately managed capabilities. The practical effect of the renaming has been consolidation of a project and control-plane experience: model catalog access, agent building and hosting, tool and knowledge connections, evaluation, and observability increasingly live inside one product surface backed by Azure identity, networking, and policy primitives, rather than a loose federation of independently configured services.

Do not assume the rename implies a single monolithic runtime. Underneath the unified experience, an enterprise deployment still touches multiple Azure resource types: a Foundry resource and project, one or more model deployments (first-party and third-party, depending on what your subscription and region expose), an agent or orchestration layer, search and data connections, and the identity, network, and monitoring resources that any production Azure workload requires. Foundry's contribution is coherence across those pieces, not the elimination of them.

MH – Applied AI recommendation: when you brief internal stakeholders, explicitly state which generation of terminology you are using and why, and add one line clarifying that "Foundry" in your documents means the current unified platform, not a specific model or a specific SDK. This single habit prevents a surprising amount of confused procurement, security review, and internal audit friction later, because reviewers frequently search for the older service names and conclude a "new" unapproved tool has appeared.

Treat the platform as a fast-moving control plane sitting on top of relatively volatile building blocks: model availability, regional footprint, quota, and specific agent or tool features change on a cadence measured in months, sometimes weeks. Anchor your internal documentation to capability categories (model access, agent runtime, evaluation, observability, identity and network integration) rather than to specific model names, specific prices, or specific regional claims, and revalidate those specifics against Microsoft Learn before every major decision point. This guide deliberately avoids stating current model counts, prices, quotas, or region lists for the same reason: they will be stale before your next planning cycle.

It also helps to be candid, internally, about what "unified" does and does not mean in practice. A unified control plane reduces how many places a platform team must go to configure identity, review a model deployment, or read a trace. It does not automatically unify your organization's data classification scheme, your security review process, or your incident response runbook. Those remain organizational artifacts that a platform can support but never substitutes for. Enterprises that treat the Foundry rename as a signal to also renew their own internal AI governance documentation tend to get more value from the consolidation than those who simply repoint existing documents at a new product name.

Finally, keep a short internal note mapping old to current terminology, updated whenever Microsoft revises naming again, because it will. A one-paragraph "here is what we used to call this, here is what we call it now, here is why nothing about our governance changed" note, attached to your landing zone documentation, resolves most of the confusion a rename otherwise creates across security review, procurement, and internal audit conversations.

2. Choosing your adoption lane

Not every enterprise should adopt Foundry the same way, and the biggest early mistake is copying a peer company's adoption pattern without checking whether the underlying constraints match. Use three lanes as a starting classification, and be honest about which one actually describes your organization today, not the one that looks best in a steering committee slide.

Lane A: Regulated or high-assurance core. Financial services, healthcare, critical infrastructure, government-adjacent, or any organization where a wrong or leaked answer creates regulatory, safety, or life-impact exposure. This lane needs private networking, strict identity boundaries, data residency discipline, and a slow, evidence-heavy path from pilot to production. Move deliberately. A single ungated production incident here can cost more than a year of "slow" adoption.

Lane B: Digitally mature enterprise, moderate risk. Most large commercial enterprises outside the strictest regulated sectors. You likely already run production Azure workloads, have an identity and network team, and can stand up a proper landing zone in weeks rather than quarters. This lane can move at a healthy pace if it resists the urge to skip governance steps because "the business wants it now."

Lane C: Exploration-stage or resource-constrained. Smaller organizations, or larger ones testing AI investment appetite before committing platform budget. This lane should treat Foundry primarily as a way to validate a business mission cheaply, with a conscious plan to either graduate into Lane B discipline or stop, rather than letting an ungoverned pilot silently become a production dependency.

Worksheet: which lane are you actually in

Score each item 0 (no), 1 (partial), or 2 (yes), for your organization today:

- A named executive owner exists for AI-enabled business outcomes, distinct from IT.
- An Azure landing zone with identity, network, and policy guardrails already exists for other workloads.
- A security and compliance function can review a generative AI workload within weeks, not quarters.
- A data governance function can tell you, workload by workload, what is allowed to reach a model.
- The organization has previously operated a production system with an explicit evaluation or quality gate.
- Finance can attribute AI workload cost to a specific business owner, not a shared IT bucket.

0-4: Lane C discipline, even if your ambition is Lane B or A. 5-8: Lane B is realistic with focused investment in the weak areas. 9-12: Lane A rigor is achievable; do not let speed pressure erode it.

Mismatched lane selection is the single most common root cause we see behind stalled Foundry programs: an organization scored a 3 on the worksheet above and tried to run a Lane A regulated rollout on Lane C foundations.

The lanes are not permanent labels. An organization can be Lane C for AI specifically while being Lane A-capable for traditional workloads, if its AI-specific governance muscle simply has not been exercised yet. The honest use of this worksheet is diagnostic, not judgmental: it tells you how much governance work to front-load before your first mission, not whether your organization is capable of eventually operating at Lane A rigor.

Lane	Typical trigger	Primary early investment	Realistic first-mission timeline
A: Regulated / high-assurance	Regulatory exposure, safety-critical or life-impact workflows	Private networking, identity rigor, compliance sign-off process	Quarters, not weeks
B: Digitally mature enterprise	Existing production Azure estate, moderate risk tolerance	Landing zone reuse, evaluation gate, operating model roles	Weeks to a small number of months
C: Exploration-stage	Budget validation, limited platform maturity	Mission discipline, honest go/no-go criteria, cost caps	Weeks, with an explicit stop option

Use the table only as an orientation aid; it does not replace the worksheet score, and an organization should not select a lane simply because its industry label matches one row. A Lane B commercial enterprise processing highly sensitive personal data on behalf of Lane A customers may still need to adopt Lane A network and identity discipline for that specific mission, even while running other missions at Lane B pace.

3. Defining the first mission

Everything downstream depends on this chapter. A "mission" here means one bounded business outcome, owned by a named accountable person, with a measurable baseline and an explicit boundary on data and actions. It is not a technology proof of concept, and it is not "explore what generative AI can do for department X."

A well-formed first mission has five properties. It has a named business owner who is not the AI platform team. It has a documented current-state baseline: how the task is done today, how long it takes, what it costs, and what quality looks like, before any AI system touches it. It has an explicit boundary describing exactly what data the system may read and what actions, if any, it may take, phrased so a security reviewer can approve or reject it without guessing. It has acceptance evidence built from real examples of the actual work, including the annoying edge cases, not a handful of friendly demo prompts. It has a production owner named before launch, meaning someone whose job explicitly includes responding to incidents and reviewing failures after the pilot excitement fades.

Favor missions that are narrow, information-dense, and reviewable by a human before consequences are large: drafting rather than sending, summarizing rather than deciding, retrieving rather than executing an irreversible transaction. Expand scope only after the narrow version has survived contact with real production traffic and a genuine evaluation gate, described in chapter 10.

Worksheet: mission definition one-pager

Fill in one page per candidate mission before writing any code or opening a Foundry project:

- Business outcome and accountable owner (name, not a team name).
- Current baseline: time, cost, quality, and volume today.
- Target improvement, stated as a range, not a guaranteed number.
- Data sources the system may read, and data it must never read.
- Actions the system may take autonomously versus actions requiring human approval.
- Failure modes that are tolerable versus failure modes that require an automatic stop.
- Who reviews evaluation results and who can veto release.
- Who owns the system in production, including nights, weekends, and vendor escalation.
- Retirement condition: what result would cause the organization to shut this down.

If any of these nine lines cannot be answered today, the mission is not ready for a Foundry project; it is still a strategy conversation. That is not a criticism, it is simply the correct sequencing, and skipping it is the most expensive shortcut available to an enterprise AI program.

4. Building the operating model before scale

A platform decision is not an operating model, and Foundry will not assign accountability for you. Before a second mission is allowed to start, decide who plays each of the following roles across the organization, and write it down somewhere more durable than a meeting note.

Platform owner. Accountable for the shared landing zone: identity patterns, network topology, policy baselines, and shared observability. This role exists once, centrally, regardless of how many missions run on top of it.

Mission owner. Accountable for one bounded business outcome, its baseline, its evaluation results, and its production health. This role exists once per mission and should not be the platform owner; conflating the two roles is a common reason platform teams end up silently owning business risk they cannot actually manage.

Security and identity owner. Approves the identity model, network boundary, and data access pattern for each mission before production traffic flows, and periodically reviews access as missions evolve.

Data owner. Approves which data sources a mission may connect to, and is accountable for classification, residency, and retention decisions that affect what reaches a model or an agent tool.

Evaluation owner. Accountable for the representative dataset, the release thresholds, and the decision to allow or block a release, independent from the engineering team building the feature; independence matters because the people who built the system are the worst-positioned to judge whether it is good enough.

FinOps owner. Accountable for cost visibility per mission, capacity planning, and catching runaway usage before the monthly bill does.

Two failure patterns dominate real deployments. The first is a "platform team owns everything" pattern, where one small central team becomes an unstated approver, operator, and risk-owner for every mission, and burns out or becomes a bottleneck within two quarters. The second is a "no one owns anything centrally" pattern, where each business unit stands up its own Foundry project with its own identity model, network exposure, and evaluation approach, producing an unmanageable sprawl of inconsistent security postures within a year. The healthy middle ground is a thin, strong central platform team that owns the landing zone and shared services, paired with a mission owner and a lightweight governance forum per business domain.

MH – Applied AI recommendation: stand up a short (60-90 minute) monthly or biweekly AI operating review that looks at every live mission's evaluation trend, cost trend, and open incidents on one page. This is far more valuable, and far cheaper, than a quarterly steering committee that reviews slideware instead of production evidence.

5. Platform and landing zone architecture

Treat your Foundry footprint the way you would treat any other Azure production landing zone: as durable infrastructure that will host many missions over its life, not as a single project folder for one pilot. Microsoft documents the platform's role and scope in What is Microsoft Foundry?, and Microsoft's own Cloud Adoption Framework guidance on AI strategy, at AI adoption strategy, is the right starting reference for sequencing platform investment against business readiness rather than the reverse.

A durable landing zone typically separates at least three concerns. First, a shared platform layer: subscription and resource group topology, identity foundations, network hub, shared policy assignments, and shared monitoring and logging destinations. Second, a per-mission project layer: a Foundry project (or projects) scoped to one mission or a closely related family of missions, with its own model deployments, agent definitions, and connected data sources, but inheriting the shared platform's identity and network guardrails rather than reinventing them. Third, an evaluation and observability layer that spans both, so that quality and cost signals can be compared consistently across missions rather than trapped inside isolated dashboards.

Resist two opposite temptations. The first is a single shared project holding every mission's models, tools, and data connections, which makes blast-radius containment and cost attribution nearly impossible once you have more than a couple of live missions. The second is a fully independent landing zone per business unit, which multiplies the security review burden and guarantees inconsistent controls. A small number of well-scoped projects, sharing one platform layer, tends to age better than either extreme.

Microsoft's Well-Architected guidance for AI workloads, at AI workloads on Azure Well-Architected Framework, is a useful architecture-review checklist to run against your landing zone design before the first mission goes live, and again whenever you add a materially different kind of mission (for example, moving from a read-only assistant to an agent that can take autonomous actions).

MH – Applied AI recommendation: design the landing zone assuming at least three to five missions will eventually run on it, even if you are only funding one today. The incremental cost of that assumption is small at design time and expensive to retrofit once a first mission's ad hoc identity and network choices have become the accidental template every subsequent team copies.

6. Identity, security, network, and compliance

This is the chapter where enterprises most often either invest correctly or quietly accumulate risk that surfaces during an audit or an incident. Four questions need explicit, written answers before production traffic flows: whose identity reaches the model and any connected data, what network path the traffic takes, what compliance and residency obligations apply, and what happens when something goes wrong.

Identity. Decide, per mission, whether the system operates using a service identity with broad access, or preserves and enforces the calling user's identity and permissions all the way through retrieval and tool calls. The second pattern is materially more work to build correctly, and materially safer for any mission touching sensitive or permissioned data; a service identity that can see everything a broad service account can see is a standing invitation to a very bad incident report. Microsoft Entra ID is the identity foundation for both patterns; treat role assignments, conditional access, and managed identities for Azure resources as mandatory, not optional hardening.

Network. Decide whether the mission's model and data traffic needs to stay off the public internet entirely. Regulated or high-sensitivity missions typically do; less sensitive internal tools may tolerate a more open path if compensating controls exist. Microsoft documents the private networking pattern for Foundry resources in Configure private link for Azure AI Foundry, which is the correct reference to work through with your network team before committing to a topology, since private networking decisions are expensive to reverse after data flows have been built around them.

Compliance and data residency. Confirm, for the actual regions and services your mission will use, what residency, retention, and regulatory commitments apply, and do not infer them from a peer company's public case study; regional service availability and compliance scope both change over time and must be re-verified for your own subscription and workload. Avoid stating specific compliance certifications about your own deployment unless a qualified internal or third-party compliance function has actually verified them for your configuration; a platform's general compliance posture does not automatically transfer to a specific customer deployment's certification status.

Incident response. Decide, before launch, what an AI-specific incident looks like for this mission (a harmful output reaching a customer, a data boundary violation, a runaway cost event, an agent taking an unintended action) and who is paged. Retrofit this after a real incident, and you will retrofit it under far worse conditions than a calm design review.

Checklist: security and network sign-off

- Identity pattern documented: service identity or delegated user identity, with justification.
- Least-privilege role assignments reviewed for the mission's service identities and managed identities.
- Network path decided: public, private link, or hybrid, with the reviewing network owner named.
- Data residency and retention obligations confirmed for the actual regions in use, not assumed from documentation examples.
- Logging and audit trail confirmed to capture who (or what identity) accessed what data, through the model or agent layer, not only through the underlying storage service.
- Incident response owner and escalation path named and tested at least once before go-live.

Treat this checklist as a gate, not a form to file after the fact. The most common way it gets diluted in practice is a well-intentioned team completing it late, after a pilot already has real users, and discovering that the honest answers to one or two lines would have changed an architecture decision made weeks earlier. Running it before the first line of integration code is written costs a day or two; running it after informal adoption has already begun can cost a full redesign, and occasionally a genuine incident.

7. Data and knowledge foundations

Generative systems are only as trustworthy as the material they can see, and most enterprises underestimate how much preparatory data work a good mission actually needs. Before connecting any data source to a Foundry project, classify it: what is public, what is internal-but-broadly-shared, what is sensitive-and-restricted, and what should never reach a model under any current mission. Write this down at the source level, not the mission level, so that the next mission does not have to re-litigate whether a given system of record is safe to expose.

For retrieval-augmented missions, quality of the retrieval layer usually matters more than the choice of model. Poorly chunked, poorly permissioned, or stale source content produces confident, fluent, wrong answers regardless of which model sits behind it. Invest specifically in three things: correct access-control propagation, so retrieval respects the same permissions the underlying source system enforces; content freshness, so the mission's owner knows how stale an answer can be before it is misleading; and traceability, so every answer can be tied back to a specific source document a human can inspect and challenge.

For missions that let an agent take actions rather than only answer questions, apply the same discipline to tools as you do to data: each tool should have an explicit, minimal scope, explicit authorization checks independent of the model's own judgment, and a clear owner who can change or revoke that tool's access without needing to redeploy the entire mission.

MH – Applied AI recommendation: maintain one living "data and tool boundary register" per mission, listing every connected source and tool, its owner, its classification, and the date it was last reviewed. This single artifact tends to be the fastest way to answer a security or audit question, and the slowest thing to reconstruct after the fact if it does not already exist.

Do not treat data preparation as a one-time setup task. Source systems change, permissions drift, and content goes stale. Schedule a recurring review, tied to the same cadence as your evaluation gate refresh described in chapter 10, so that data boundary decisions and evaluation evidence age together rather than silently diverging.

8. Model strategy: choice, portfolio, and exit

Microsoft Foundry's model catalog is designed to make multiple first-party and third-party models available behind a broadly consistent access pattern; Microsoft describes the current model landscape and access model in Foundry Models overview. Treat that catalog as a portfolio to select from deliberately, not a single default to accept without comparison, and treat any specific model name you choose today as a component you will likely replace, not a permanent architectural commitment.

Build model selection around the mission's actual requirements rather than general reputation: required task quality on your own representative evaluation set (chapter 10), acceptable latency for the interaction pattern, cost per unit of work at expected volume, data handling and regional fit, and the operational maturity of the specific deployment option (managed, serverless, or otherwise) in your chosen region. A model that scores well on public benchmarks can still be the wrong choice if it fails your own representative cases, costs more than your mission can sustain at real volume, or is not available in the region your compliance boundary requires.

Plan for model change from day one. Version your prompts, your evaluation dataset, and your deployed model together, so that a model update, a deprecation, or a deliberate switch to a different model can be evaluated as a controlled change rather than discovered as a production regression. Avoid hard-coding assumptions about a specific model's behavior deep inside application logic; where possible, isolate model-specific prompt engineering behind an interface your evaluation harness can swap without touching the rest of the mission.

MH – Applied AI recommendation: treat "which model" as the least strategic decision in the program and "how do we prove and re-prove that our chosen model still meets the mission's bar" as the most strategic one. Enterprises that build strong evaluation muscle can absorb model churn calmly; enterprises that picked one model and never built that muscle experience every model update as a fire drill.

Do not assume Foundry is the right home for every model workload. If a mission needs a model running fully offline, in a disconnected environment, or under constraints that private cloud networking cannot satisfy, a local or edge deployment pattern may be more appropriate than a cloud-hosted Foundry deployment; evaluate that option honestly rather than forcing every workload into one platform because the platform is already in place for other missions.

Worksheet: model selection criteria, weighted by mission

Score each candidate model against these criteria for a specific mission, rather than in the abstract; a model that wins on one mission's weighting can lose on another's:

- Task accuracy on your own representative evaluation set, not a public leaderboard result.
- Latency under realistic concurrent load for the mission's interaction pattern.
- Cost per successful task outcome at projected production volume, not cost per call in isolation.
- Data handling fit: whether the deployment option and region satisfy the mission's residency and network requirements.
- Operational maturity of the specific deployment option in your target region, including support and update posture.
- Portability: how much application logic would need to change if this model were replaced.

Weight these differently per mission. A customer-facing latency-sensitive assistant should weight latency and cost per outcome heavily; a back-office analytical workflow processing sensitive records overnight can weight data handling fit and task accuracy more heavily than latency. Resist a single enterprise-wide model policy that ignores these differences; it tends to either overpay for missions that did not need the strongest available model, or underserve missions that genuinely did.

9. Agents and tools: from prompt to acting system

An agent, in this platform's vocabulary, is a system that can plan, call tools, and take multi-step action toward a goal, rather than simply answering a single prompt. Microsoft describes the current agent capability and its production controls in Foundry Agent Service overview. Moving from a single-turn assistant to an acting agent is a significant increase in both value and risk, and it deserves a correspondingly higher bar of review, not the same sign-off you used for a read-only chat assistant.

Before granting an agent any tool that changes state (sends an email, updates a record, moves money, opens a ticket, calls another system), require three things: an explicit allow-list of the exact actions the tool can perform, with no implicit broader scope than documented; an authorization check that happens independently of the model's own reasoning, so a manipulated or mistaken model output cannot itself grant permission it should not have; and a human approval point for actions above an agreed risk or reversibility threshold, at least until the mission has enough production evidence to justify full autonomy for that specific action.

Design for a graceful degradation path. An agent that cannot complete a task should say so and hand off to a human, not silently guess or retry indefinitely. Log every tool call, its arguments, its result, and the reasoning trace that led to it, so that a failure can be diagnosed from evidence rather than reconstructed from memory. Treat multi-agent designs, where several specialized agents coordinate, as a further increase in complexity and review burden; they can be powerful, but they multiply the surface area for miscommunication between agents in ways that are genuinely harder to test than a single agent's behavior.

Checklist: before an agent gets a new tool

- The tool's scope is written down as an explicit allow-list, not inferred from a broad API credential.
- Authorization is checked outside the model's own output, using a mechanism the model cannot talk its way around.
- Every call, its arguments, and its result are logged and traceable to a specific mission run.
- A human approval step exists for actions above the mission's agreed risk threshold.
- A rollback or compensating action exists for anything the tool can do that is not trivially reversible.
- The tool's owner can revoke or change its access without redeploying the whole mission.

10. The evaluation gate

Evaluation is the single highest-leverage discipline in this entire handbook, because it is the mechanism that turns "it looked good in the demo" into an actual release decision. Microsoft's observability documentation, at Observability in Generative AI, describes the platform's evaluators, tracing, and monitoring capabilities; the organization still has to decide, for its own mission, what "good enough to ship" actually means.

Start from the operating claim the mission depends on, stated as a single sentence: given this kind of input, the system should produce this kind of output, which a qualified reviewer can accept or correct without discovering unsupported facts or unauthorized actions. That sentence tells you exactly what evidence you need: representative inputs, expected use of sources, forbidden claims, acceptable reviewer effort, acceptable latency, and the treatment of uncertainty.

Build a representative evaluation dataset from real work, not convenient examples. Include the frequent, commercially important cases; the difficult but valid edge cases; incomplete and contradictory inputs; cases that should be refused or escalated rather than answered; sensitive data and authorization boundary cases; tool and dependency failure cases; and any failure the organization has already observed in production. Version this dataset explicitly and protect it with the same care as production data, since it typically contains real examples of sensitive work.

Combine several kinds of measurement rather than trusting one score. Deterministic checks verify schema conformance, citation presence, required fields, and exact tool arguments. Task measures assess whether the system actually completed the business job, which is often mission-specific and may require expert human review rather than an automated score. Quality measures assess relevance, groundedness, and coherence, but must never substitute for task correctness; a fluent, well-structured wrong answer is still wrong. Safety and security measures test harmful content, prompt injection resistance, and sensitive data handling. Agent measures examine tool-call accuracy, trajectory quality, and whether the system stayed inside its allowed operating envelope. Treat LLM-as-judge evaluators as useful but probabilistic instruments; calibrate them against human judgment on a sample and inspect disagreements rather than trusting the automated score blindly.

Define the release gate's pass criteria before you run the final evaluation, not after you see the results. A reasonable gate requires zero critical authorization or data-boundary failures, all required deterministic checks passing, a minimum task-success threshold on the priority cases, no unacceptable regression against the previous version, acceptable latency and cost under realistic load, and named acceptance of any known residual failure by someone with the authority to accept that risk. Report the full distribution of results and the specific failing cases, not only an average; a 95 percent aggregate score can hide a catastrophic failure concentrated in exactly the five percent of cases that matter most to your business or your regulator.

MH – Applied AI recommendation: assign evaluation ownership to someone independent of the engineering team that built the mission. The people who built the system are structurally the worst-positioned judges of whether it is good enough, not because they lack integrity, but because they have spent weeks unconsciously calibrating to the system's known weaknesses.

11. GenAIOps and observability in production

Passing an evaluation gate once is necessary but not sufficient; production is where a mission's real distribution of inputs, failures, and drift appears. GenAIOps, the operational discipline around generative AI systems, extends familiar DevOps and MLOps practice with a few AI-specific additions: distributed traces that capture model calls, retrieval steps, and tool invocations across a multi-step flow; continuous, not only pre-release, evaluation of sampled production traffic within your privacy and retention boundaries; and version lineage that ties every production run back to a specific application, prompt, agent, model, tool, and dataset version, because without that lineage a regression is nearly impossible to diagnose.

Use traces to answer "why" a result was wrong, not only "that" it was wrong. A bad answer might trace back to retrieval returning the wrong source, a tool receiving a malformed argument, or an instruction change that quietly altered the decision path several steps earlier. Alert on conditions that matter to the business owner and the mission's operating claim, such as a rising refusal rate, a rising authorization-failure rate, or a latency regression on the priority case types, rather than alerting on every statistical fluctuation, which trains the operating team to ignore alerts entirely.

Feed confirmed production incidents and recurring failure patterns back into the evaluation dataset from chapter 10, so the gate gets stronger over time instead of staying frozen at its launch-day state. Re-run the full evaluation gate whenever prompts, models, retrieval configuration, tools, policies, or application logic change materially, not only when the underlying model provider announces an update; many regressions originate in the application layer, not the model.

MH – Applied AI recommendation: put evaluation trend, cost trend, and open incident count for every live mission on one shared operating page, reviewed on the cadence described in chapter 4. A mission that cannot produce that page on demand is not actually being operated; it is being hoped for.

Distinguish observability from mere logging. Logging tells you a call happened; observability lets you reconstruct why a specific outcome occurred, compare it against prior versions, and connect it to the business claim the mission depends on. A mission with extensive logs but no way to answer "why did this specific case fail, and has failure like this happened before" has not actually achieved GenAIOps maturity, regardless of how much data it retains.

12. Cost, capacity, and FinOps for AI workloads

Generative AI cost behaves differently from most traditional application cost: it scales with usage in ways that are easy to underestimate during a low-traffic pilot and easy to lose control of once a mission becomes popular internally. Treat cost and capacity planning as a recurring engineering discipline, not a one-time estimate produced before launch and never revisited.

Track cost per mission, not only per subscription, so that a business owner can see the actual cost of the outcome they are accountable for, and so finance can attribute spend correctly instead of absorbing it into a shared AI budget line that obscures which missions are actually worth their cost. Track the components separately where possible: model inference cost, any retrieval or search infrastructure cost, agent and tool execution cost, and the surrounding compute, storage, and networking cost of the landing zone itself.

Set explicit budget alerts and, for missions where a runaway loop is plausible (an agent retrying indefinitely, a tool calling itself in a cycle, a caching failure causing repeated expensive calls), set hard usage caps, not only soft alerts that someone might notice after the damage is done. Revisit capacity assumptions whenever a mission's usage pattern changes meaningfully, such as a rollout to a new business unit or a new integration that multiplies call volume.

Avoid two common cost mistakes. The first is optimizing cost before proving value, which leads teams to select a cheaper, weaker model that fails the evaluation gate and produces a worse business outcome than a slightly more expensive model would have. The second is never revisiting cost after launch, which allows a mission that made sense at pilot volume to become quietly unsustainable at production volume, discovered only when finance escalates a bill.

Worksheet: FinOps readiness for a mission

- Cost is attributable to this specific mission, separate from other missions sharing the platform.
- A budget owner has approved an expected monthly range, not just an initial pilot estimate.
- A hard usage cap or circuit breaker exists for any plausible runaway-cost scenario.
- Cost per successful task outcome (not just cost per API call) is tracked, so efficiency work targets the right thing.
- Capacity assumptions are revisited on a defined cadence, not only when a bill surprises someone.

A useful habit is to separate cost reporting into three horizons, because each answers a different question for a different audience. A daily or weekly operational view catches runaway loops and anomalies quickly enough to intervene before they become a material incident; this is the platform and mission owners' view. A monthly attribution view rolls cost up per mission and compares it against the budget range approved for that mission, feeding the FinOps owner's regular reporting. A quarterly capacity-planning view looks at trend, not snapshot, and asks whether the mission's growth trajectory will outrun its current budget envelope before the next planning

cycle, feeding the executive decision pack described in chapter 16. Missions that only produce the first view tend to be surprised by their own success; missions that only produce the third view tend to discover runaway cost far too late to intervene cheaply.

13. A realistic 90-day path

A credible 90-day plan for a first mission has three phases, each with a clear exit condition, and it explicitly plans for the possibility that the honest answer at any phase is “stop,” which is a legitimate and valuable outcome, not a failure of the program.

Days 1-30: prove the task. Confirm the mission one-pager from chapter 3 is complete and signed off by its business owner. Stand up the minimum landing zone slice needed for this mission (identity, network boundary, logging), not a fully generalized platform. Build a first working version against real, representative examples, including the difficult ones, and honestly assess whether the underlying task is even solvable well enough to be worth continuing. Exit condition: a documented go or no-go decision from the mission owner, based on real examples, not a friendly demo.

Days 31-60: prove the controls. Build the representative evaluation dataset and run the full evaluation portfolio from chapter 10. Test permissions, data boundaries, refusal behavior, and escalation paths deliberately, including adversarial inputs designed to break them. Test latency and cost under realistic, not best-case, load. Complete the security and network sign-off checklist from chapter 6. Exit condition: the mission passes its own defined release gate, with any residual known failures explicitly accepted by a named owner with the authority to accept that risk.

Days 61-90: prove the operation. Deploy to production for the real user population, with tracing, monitoring, and the operating review cadence from chapter 4 already running, not scheduled to start later. Confirm the production owner named in chapter 3 is actually receiving alerts and reviewing failed traces, not merely listed on a document. Run at least one full evaluation-gate re-run against a deliberate change (a prompt tweak, a model version bump, a tool change) to confirm the change-control loop actually works under real conditions. Exit condition: a documented decision on whether this mission is ready to be a template for the next one, described in chapter 17.

Do not compress this sequence to satisfy an external deadline; compressing phase two, the controls phase, is the most common way an enterprise turns an otherwise reasonable pilot into a production incident.

14. Common failure patterns

These patterns recur across enterprises adopting Foundry, independent of industry, and recognizing them early is cheaper than discovering them in an incident review.

The platform-first launch. A Foundry project is created because a platform initiative needs a visible artifact, before any specific business mission has been defined. Fix: require a completed mission one-pager, per chapter 3, before any Foundry project is provisioned for a new mission.

The broad-access pilot. A pilot is connected to a wide set of internal data "to see what is useful," without a data boundary register, because narrowing the connection felt like it would slow the demo down. Fix: require the data and tool boundary register from chapter 7 before any data source is connected, even for an early pilot.

The friendly-demo evaluation. The only evidence of quality is a handful of cooperative example prompts run by the project team itself. Fix: require the representative evaluation dataset and release gate from chapter 10 before any production rollout, with an evaluation owner independent from the engineering team.

The orphaned pilot. A pilot succeeds, gets used informally by more people than intended, and has no named production owner when something eventually goes wrong. Fix: no mission proceeds past day 30 of the plan in chapter 13 without a named production owner who has accepted the role.

The silent cost creep. Usage grows organically, no one is tracking cost per mission, and a large bill surfaces the problem months after it started. Fix: the FinOps worksheet in chapter 12, reviewed on the same cadence as the operating review in chapter 4.

The frozen evaluation set. An evaluation dataset is built once at launch and never updated, so it stops reflecting the mission's actual current failure modes. Fix: feed confirmed incidents back into the dataset, per chapter 11, on a defined recurring cadence.

The over-scoped agent. An agent is given a broad tool credential "for flexibility," rather than an explicit minimal allow-list, and a manipulated or mistaken output causes an action no one intended to authorize. Fix: the tool checklist in chapter 9, applied before any state-changing tool is granted, with no exceptions for "just this once."

The single point of platform ownership. One central team becomes the unstated approver, operator, and risk owner for every mission across the enterprise, and becomes both a bottleneck and a single point of institutional risk. Fix: the role separation in chapter 4, with a genuinely independent mission owner for each business outcome.

15. The enterprise adoption scorecard

Use this scorecard quarterly, across every live mission, to get an honest, comparable view of program health rather than relying on anecdote or the loudest stakeholder's opinion. Score each dimension 0 (absent), 1 (partial), or 2 (solid and evidenced) for each mission, and total the six dimensions for a score out of 12.

Dimension	0: Absent	1: Partial	2: Solid
Mission ownership	No named business owner	Named but not accountable for outcomes	Named, accountable, reviews evidence
Identity and network	Broad service identity, open network	Partial isolation, inconsistent review	Least-privilege identity, reviewed network path
Data and tool boundary	No boundary register	Register exists, not maintained	Register exists and is reviewed on a cadence
Evaluation gate	Demo-only evidence	Dataset exists, gate informal	Representative dataset, explicit thresholds, independent owner
Observability and operations	No tracing or alerts	Tracing exists, no owner reviews it	Tracing, alerts, and a named owner reviewing regularly
Cost and capacity	No mission-level cost visibility	Cost visible, no caps or alerts	Cost attributed, capped, and reviewed on a cadence

A total of 0-4 signals a mission that should not be in production yet, regardless of how well it performs functionally. A total of 5-8 signals a mission that is operating with real, named gaps that should be closed on an explicit timeline. A total of 9-12 signals a mission that is a legitimate candidate to become a template for the next one, per chapter 17.

16. The executive decision pack

When you need to secure budget, headcount, or executive sponsorship for continued Foundry investment, resist the temptation to lead with platform capability. Lead with mission evidence. A credible decision pack contains: the mission's baseline and measured outcome, stated honestly including where results fell short of hope; the evaluation gate results, including known residual failures and who accepted that risk; the cost per successful outcome at current volume, and a capacity projection for the next planning horizon; the scorecard from chapter 15 for this mission and, if applicable, for the portfolio of live missions; a clear statement of what is reusable for the next mission (landing zone, identity pattern, evaluation harness) versus what was specific to this one; and a named recommendation for what should happen next, including "stop" as a legitimate option when the evidence supports it.

Avoid two weaknesses that undermine executive credibility. The first is presenting only favorable metrics while omitting known failure modes; a reviewer who later discovers an omitted failure will discount every future report from that team, which costs far more than the discomfort of disclosing it now. The second is presenting a platform capability narrative ("we now have an enterprise AI platform") in place of a business outcome narrative ("this specific process now takes X percent less time, verified against Y real cases, with Z known limitations"); executives approving further investment need the second, not the first.

MH – Applied AI recommendation: keep the decision pack to a small number of pages built entirely from evidence already produced during the 90-day path in chapter 13. If producing the pack requires new work beyond formatting existing evidence, that is itself a signal the mission's operating discipline has gaps worth closing before asking for more investment.

17. Choosing the next mission

The first mission's real deliverable is not just its own business outcome; it is a tested template for everything reusable in chapters 5 through 12: the landing zone, the identity and network pattern, the data boundary process, the evaluation harness, and the observability and cost practices. Before selecting a second mission, explicitly separate what is reusable from what was specific to mission one, and document that separation so the second mission's team does not have to rediscover it by trial and error.

Choose the next mission using the same discipline as the first: a bounded outcome, a named owner, a documented baseline, and an explicit data and action boundary, evaluated against the lane assessment from chapter 2 and the scorecard from chapter 15. Resist the temptation to select the next mission purely because a business unit is enthusiastic; enthusiasm without a measurable baseline reproduces the platform-first failure pattern from chapter 14 under a different name.

Deliberately vary the second mission's shape from the first, if possible: if mission one was a read-only assistant, consider a retrieval-heavy or agentic mission next, so the reusable platform gets tested against a genuinely different risk profile rather than only a similar one. This surfaces gaps in the landing zone and governance model while the stakes are still manageable, rather than after a much larger rollout has been built on an under-tested foundation.

Expect the second and third missions to move faster than the first, because the landing zone, identity model, and evaluation harness are now proven rather than theoretical. If they are not moving meaningfully faster, that is a signal the reusable foundation was not actually reusable, and it deserves an honest post-mortem before a fourth mission compounds the same gap.

18. Glossary

Microsoft Foundry. Microsoft's current unified platform name for building, evaluating, deploying, and operating generative AI applications and agents on Azure; successor terminology to Azure AI Foundry and, before that, Azure AI Studio.

Azure AI Foundry / Azure AI Studio. Earlier names for substantially the same conceptual platform space; still common in older documentation, code, and organizational vocabulary.

Landing zone. The shared platform layer of identity, network, policy, and monitoring infrastructure that hosts one or more missions, distinct from any single mission's own project resources.

Mission. A bounded business outcome with a named owner, a documented baseline, an explicit data and action boundary, and a defined production owner, as distinct from a general technology proof of concept.

Agent. A system that can plan, call tools, and take multi-step action toward a goal, rather than only answering a single prompt.

Tool. A discrete capability an agent can invoke, ideally scoped to an explicit, minimal allow-list with independent authorization checks.

Evaluation gate. A release decision point requiring representative test evidence and explicit pass thresholds before a mission may reach or remain in production.

GenAIOps. The operational discipline extending DevOps and MLOps practice to generative AI systems, including tracing, continuous evaluation, and version lineage across prompts, models, and agents.

FinOps. The operating discipline of attributing, forecasting, and controlling cloud and AI workload cost by business owner and outcome, rather than treating it as an undifferentiated shared expense.

Private link / private networking. A network pattern that keeps traffic between a workload and its Azure resources off the public internet, typically required for regulated or high-sensitivity missions.

LLM-as-judge. The practice of using a language model to score another system's outputs against defined criteria; useful but probabilistic, and best calibrated against human judgment rather than trusted unconditionally.

Groundedness. The degree to which a system's output is supported by the source material it was given, rather than invented; a common quality measure in an evaluation portfolio.

Trace. A recorded, step-by-step record of a single run through a model, retrieval, or agent flow, used to diagnose why a specific result occurred.

Content safety. A category of checks and filters aimed at detecting and mitigating harmful, unsafe, or policy-violating content in model inputs or outputs.

Foundry project. The per-mission (or per-mission-family) workspace within a Foundry resource that scopes model deployments, agent definitions, and connected data and tools for a specific piece of work.

19. References and methodology

Primary Microsoft references used in this guide

- Microsoft, What is Microsoft Foundry?
- Microsoft, Navigate from Foundry classic
- Microsoft, Foundry Control Plane overview
- Microsoft, AI adoption strategy, Cloud Adoption Framework
- Microsoft, AI workloads on Azure, Well-Architected Framework
- Microsoft, Foundry Agent Service overview
- Microsoft, Foundry Models overview
- Microsoft, Configure private link for Azure AI Foundry
- Microsoft, Role-based access control for Microsoft Foundry
- Microsoft, Authentication and authorization in Microsoft Foundry
- Microsoft, Responsible AI in Microsoft Foundry
- Microsoft, Observability in Generative AI
- Microsoft, Agent evaluators
- Microsoft, Manage costs for Microsoft Foundry
- Microsoft, Data, privacy, and security for models sold directly by Azure

Methodology

This guide was written by MH – Applied AI as an independent, vendor-neutral field manual, drawing on Microsoft's own published documentation for product facts and on our own operating experience for the recommendations, checklists, and failure patterns, which are labeled as MH – Applied AI recommendations throughout. Product facts are cited to their canonical Microsoft Learn source wherever possible; recommendations reflect our judgment and are open to disagreement, not claims of a single correct answer.

We deliberately avoided stating specific model names, prices, quotas, or regional availability, because these details change frequently and any figure printed here would likely be stale before you finish reading it. Before making a budget, procurement, security, or architecture decision based on any specific capability, always verify current details directly against Microsoft Learn and your own Azure subscription, since availability, pricing, and features vary by region, subscription type, and time.

Limitations

This guide does not constitute legal, regulatory, security, or compliance advice, and it makes no claim of certification, guaranteed return on investment, or specific client outcomes. It does not reflect a Microsoft partnership, sponsorship, or endorsement of any kind. Organizational maturity, regulatory context, and risk appetite vary widely; treat every worksheet, checklist, and threshold in this guide as a starting point to adapt, not a fixed standard to apply unchanged.

Source review date

Content and references last reviewed for currency on 2026-07-28. Given the pace of change in this space, we recommend re-verifying every cited Microsoft Learn reference, and any specific capability claim you plan to rely on, no less often than quarterly, and immediately before any major architecture or investment decision.

Who this handbook is for

This handbook is written for enterprise executives sponsoring AI investment, transformation leaders responsible for turning that investment into working systems, and the architects and platform engineers who will actually configure identity, networking, evaluation, and observability around Microsoft Foundry. It assumes familiarity with enterprise Azure operations and standard cloud governance vocabulary, but not deep prior generative AI experience. Readers new to the space should start at chapter 1 and move sequentially; readers already running a Foundry pilot may prefer to start at chapter 15's scorecard, use it to identify the weakest dimension of their current program, and jump directly to the corresponding chapter for remediation guidance.

MH – Applied AI AI, deployed where your business runs. +41 76 390 23 17