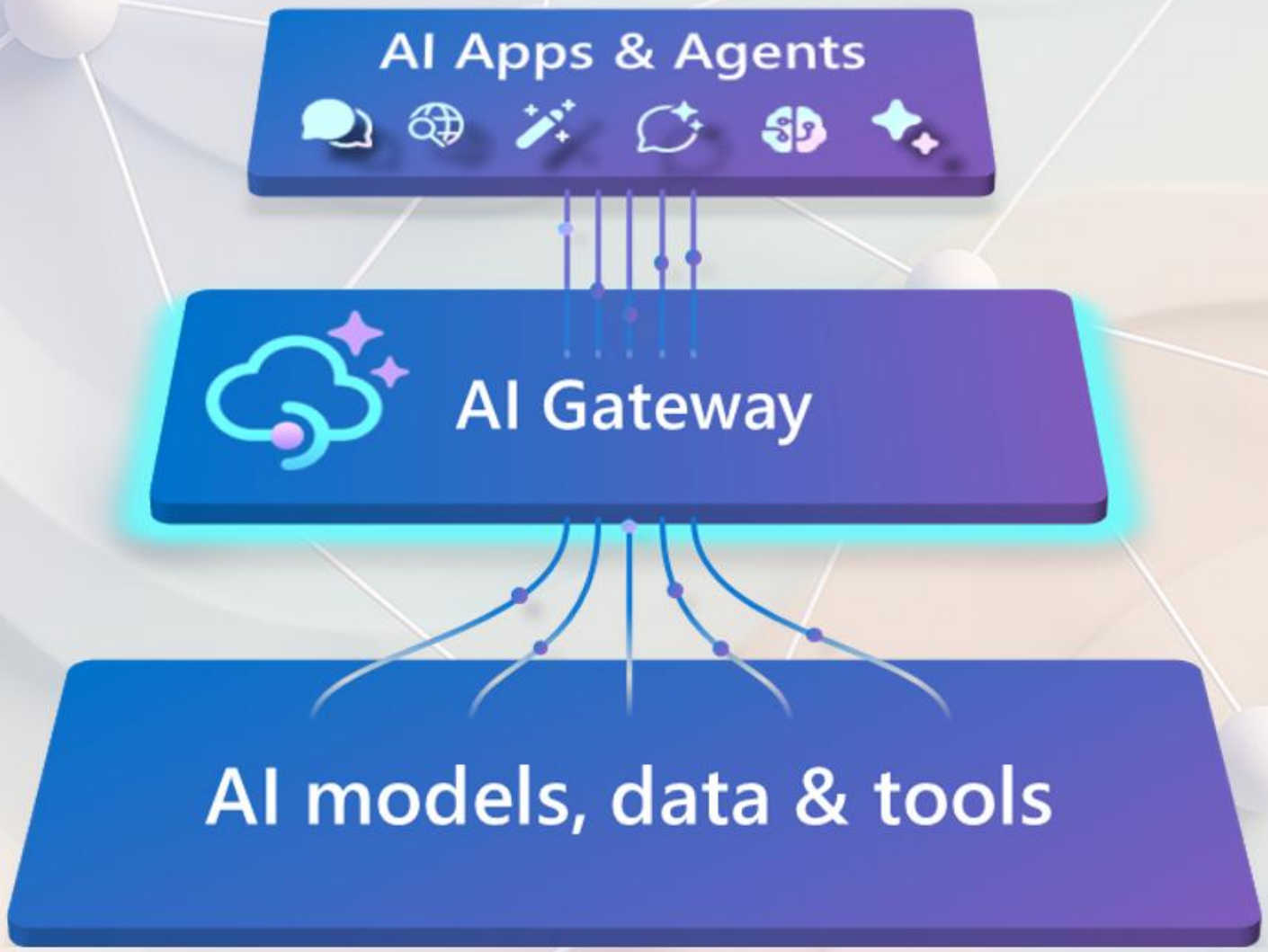
Enterprise AI Gateway

An in-depth exploration on using Azure API Management and Microsoft Foundry to secure, observe and control AI Apps & Agents



//Alex Vieira

Microsoft

This book, written in August 2025 and updated in February 2026, is provided "as-is" and reflects the author's knowledge and views at the time of writing; its content and referenced information may change without notice.

Acknowledgments

Working with customers around the world has given me valuable insights into their evolving needs and AI challenges, inspiring me to write this book to help readers use the AI gateway pattern safely and effectively in enterprises.

Huge thanks to the Azure API Management product team, the Global Black Belt teammates, Solution Engineers, and Cloud Solution Architects at Microsoft for their help in writing this book.

About the Author

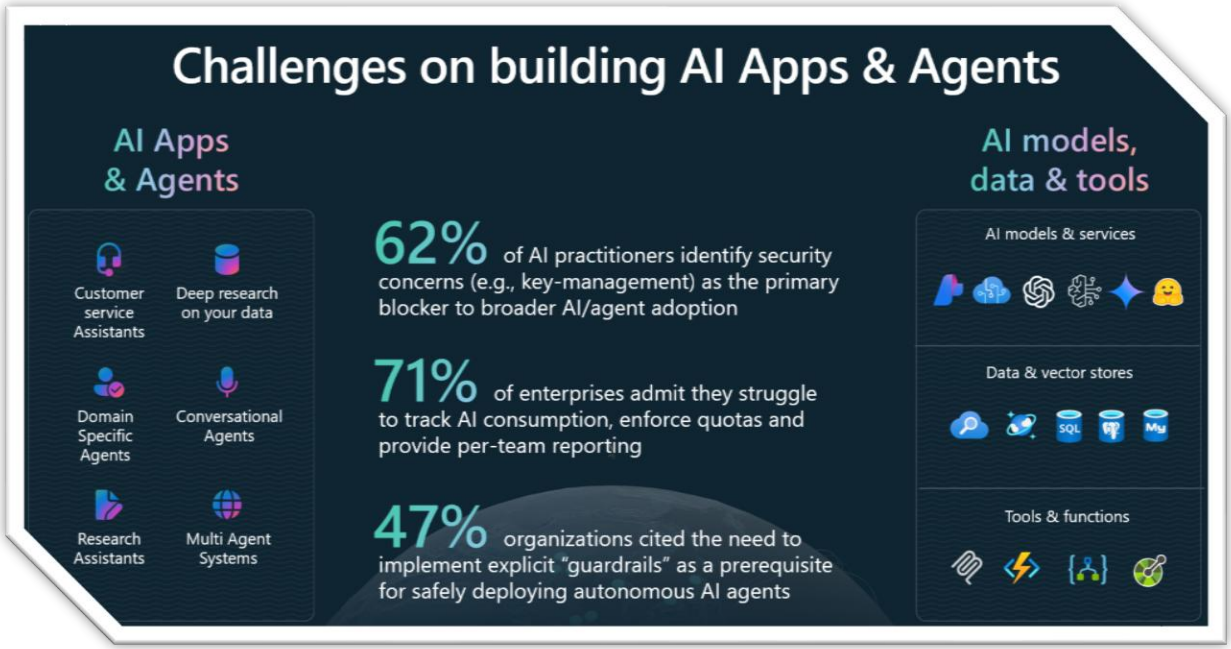
Alex Vieira, a Senior Global Black Belt at Microsoft, specializes in designing and governing secure, scalable AI applications and agents. He enables organizations to implement practical AI solutions by translating emerging AI patterns into real-world architectures and providing hands-on guidance.

Maintains the AI Gateway repository (http://aka.ms/ai-gateway), which provides labs for experimenting with various AI patterns to enhance security, observability, and control.

Contents

Chapter 1: Introduction to the AI Gateway Pattern

Enterprises adopting AI at scale face a critical need for a centralized component to govern and secure AI usage across the organization.



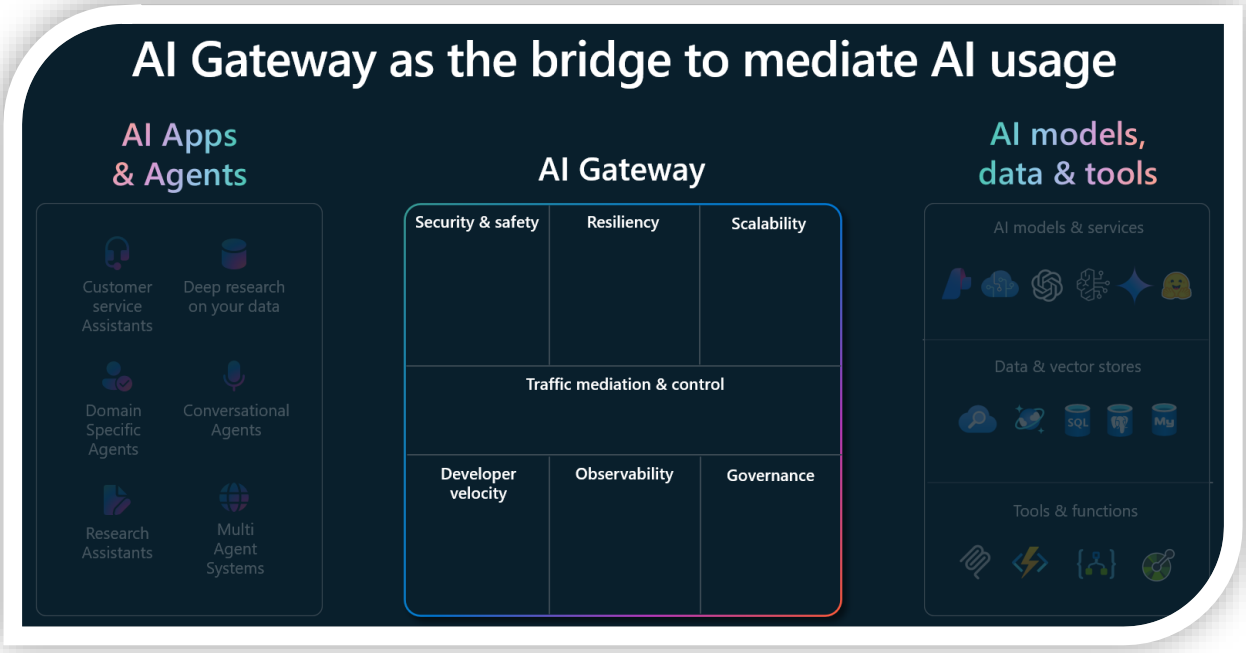
The AI Gateway is that conceptual component that sits between AI consumers (applications, agents, users) and AI providers (models, data sources, tools), enforcing policies and providing oversight. Without such a gateway, organizations encounter significant challenges and risks when consuming model APIs, data, and tools directly:

- **Uncontrolled Usage & Costs:** If multiple applications call AI services without coordination, one app could exhaust shared resources or budget, leaving others starved. Teams struggle to track who is using what and to enforce fair quotas or cost allocation. This can lead to surprise overruns and inability to charge costs back to the responsible units.
- **Security & Compliance Risks:** Direct access to AI models often means embedding API keys in many applications, increasing the risk of key leakage or misuse. Without an AI Gateway, it's challenging to ensure every AI call complies with security policies (authentication/authorization standards, data handling

rules). Sensitive data might be sent to external models without proper checks, raising compliance and privacy concerns.

- **Lack of Observability:** Organizations have little visibility into AI usage patterns, prompts, or outputs if calls bypass central logging. This hampers debugging, auditing, and responsible AI behavior. For example, if a user gets an inappropriate response from a third-party model, it's difficult to trace the prompt or response content without a centralized log.
- **Inconsistent Governance:** Each team might integrate AI differently, duplicating effort and creating inconsistent safeguards. There is no unified way to enforce **"guardrails"** (rules on AI behavior and usage) across all AI applications. Some applications might lack crucial measures like rate limiting or content moderation, leading to uneven risk management.
- **Operational Complexity:** Scaling AI adoption without a gateway means *every* new model or tool integration requires custom development for logging, security, rate limiting, etc. This slows down innovation and leads to brittle ad-hoc solutions. Each project reimplements the same integration logic (error handling, retries, caching), resulting in inefficiency and potential errors.

An **AI Gateway pattern** addresses these issues by providing a managed layer through which all AI consumption flows.



It acts as a **secure proxy** for AI services so that applications never directly call models or tools with raw credentials – the gateway manages authorization and access on their behalf. It offers **policy enforcement** for usage limits, content filtering, and routing to ensure fair and safe use of AI resources. It also provides **monitoring and logging** for all AI interactions, giving enterprises insight into how AI is used, by whom, and at what cost.

The **table** below summarizes the risks of the "direct access" approach versus the benefits of using an AI Gateway:

AI Consumption: Direct Access vs. AI Gateway Approach

Topic	Without an AI Gateway (Direct Access)	With an Enterprise AI Gateway
Security	Apps embed and distribute API keys for AI services, resulting in higher risk of leaks or misuse. No central control of authentication per call.	Keyless access via managed identities where the gateway authenticates on behalf of apps (no keys in client code). Centralized auth (OAuth, tokens) and consistent zero-trust policy enforcement.
Usage Control	Each app consumes AI independently. One application can monopolize model throughput (tokens/minute), starving others. Hard to enforce org-wide quotas or allocate costs per team.	Token rate limiting and quotas per consumer ensure fair share of model usage. Gateway tracks usage per app/team, enabling chargeback and preventing any single client from exhausting resources.
Observability	Limited insight into prompts or responses. Logging and monitoring must be implemented by each team (if at all), leading to blind spots in AI usage.	Central logging of requests, responses, and token counts for all AI calls. Built-in dashboards provide usage trends. Unified logs enable auditing prompts and outputs for compliance and debugging.
Governance	No uniform way to enforce content standards or usage policies. Potential for regulatory or ethical violations (e.g. unsafe outputs) goes unchecked.	Content safety checks and policies applied uniformly (e.g. block disallowed prompts via integration with content moderation). Only approved models/tools are accessible. Organizational guardrails in place by default.

Integration Effort	Redundant work as each project integrates AI services and handles issues (auth, errors) on its own. Inconsistent interfaces for tools/data access.	Standardized integration layer: one gateway for all models and tools means a consistent API for developers. Faster onboarding of new AI capabilities with pre-built policies and connectors. Existing enterprise APIs can be exposed as AI tools with minimal effort (via standardized protocols like MCP).
Resilience & Scale	Each integration must handle failures or scale issues (if a model endpoint fails or hits capacity). Limited ability to load balance across model instances.	Resilient routing and scaling: gateway can load balance across multiple model endpoints and apply circuit breakers. It can prioritize using dedicated capacity (PTUs) before overflowing to pay-as-you-go instances and support multi-regional deployments for global availability.

In essence, the AI Gateway pattern is emerging as an **industry best practice for enterprise AI adoption**, ensuring that AI's power is harnessed with appropriate oversight. Analyst discussions and early adopters echo this need: *"As AI becomes more deeply integrated into applications, managing and governing AI APIs is more important than ever."* Enterprises have found that imposing a gateway improves their ability to apply **security, compliance, and cost controls** on AI services.

In the next chapters, we dive deeper into how to build such an AI Gateway using Microsoft's technologies (Azure API Management, Microsoft Foundry and other Azure services), and how it governs AI **Models**, **Data**, and **Tools**. We'll explore the gateway's capabilities (traffic control, security, scaling, etc.), and provide guidance on implementation, operations, and future trends (like governing AI agents). By the end, it will be clear why an AI Gateway is a **fundamental component** of any enterprise AI architecture – enabling organizations to innovate with AI *confidently and responsibly*.

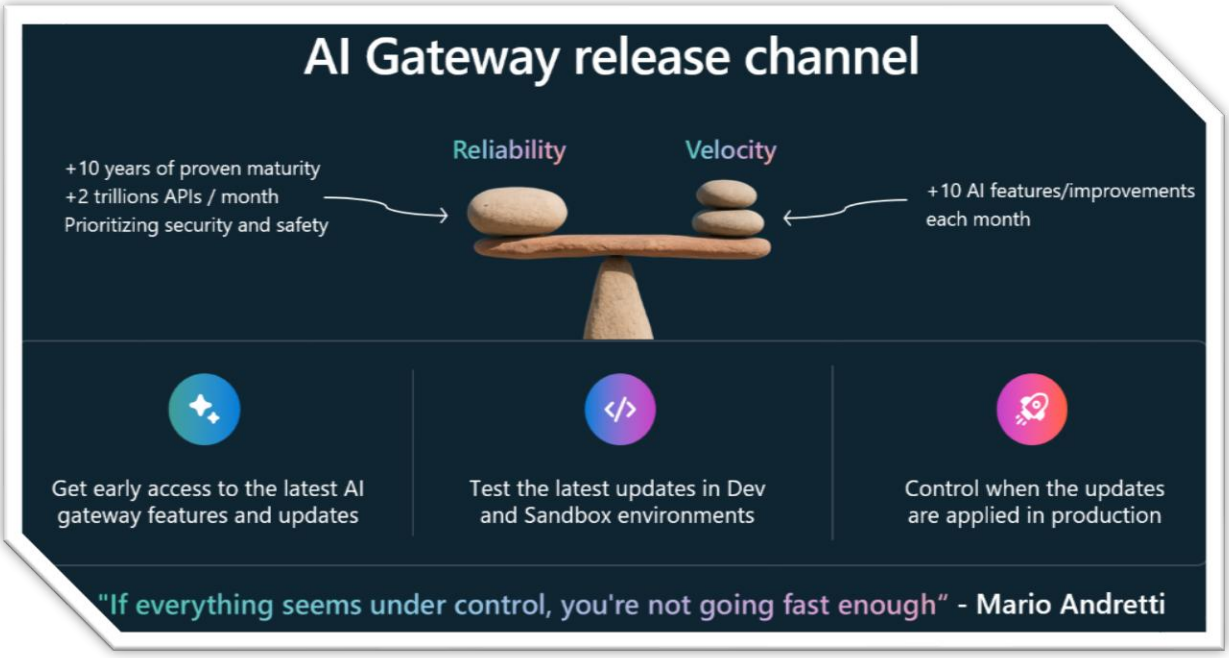
Chapter 2: AI Gateway Foundation – Azure API Management & Microsoft Foundry

Azure API Management (APIM) and Microsoft Foundry are better together to form the foundation of the Enterprise AI Gateway. Azure API Management is a mature platform for publishing, securing, and managing APIs at scale — now augmented with specialized "AI gateway" capabilities to handle AI model endpoints and MCP servers. Microsoft Foundry is a platform for designing, customizing, and managing AI applications and agents, providing a model catalog with a unified **Azure AI Model Inference API** for deploying and serving models, model evaluation, agent factory and much more. Together, these services allow enterprises to expose AI models, data and tools as managed APIs, adding critical governance layers in the process.

At its core, the AI Gateway uses **API Management as the entry point** through which AI apps and agents consume model APIs. API Management acts as a **policy enforcement point** and proxy, while Foundry (and related AI services like Azure OpenAI) act as the backend where the models actually run. By importing Foundry's model endpoints into APIM, we effectively "wrap" those models with enterprise-grade features such as authentication, rate limiting, caching, logging, and more.

How Azure API Management and Foundry Work Together: When you deploy a model in Microsoft Foundry (or in Azure OpenAI Service), you get a REST endpoint to invoke that model (for example, a URL for a chat completions API). With one-click integration, APIM can **import this AI endpoint as a new API** on the gateway. The APIM instance is often configured in the same virtual network or environment as the Foundry instance for secure connectivity. APIM sets up a **backend** referencing the Foundry endpoint (optionally via private networking) and binds a managed identity with permission to call that endpoint. From then on, applications will call **the APIM gateway's URL (not the Foundry URL)** to use the model, and APIM will forward requests to Microsoft Foundry **after applying any policies**.

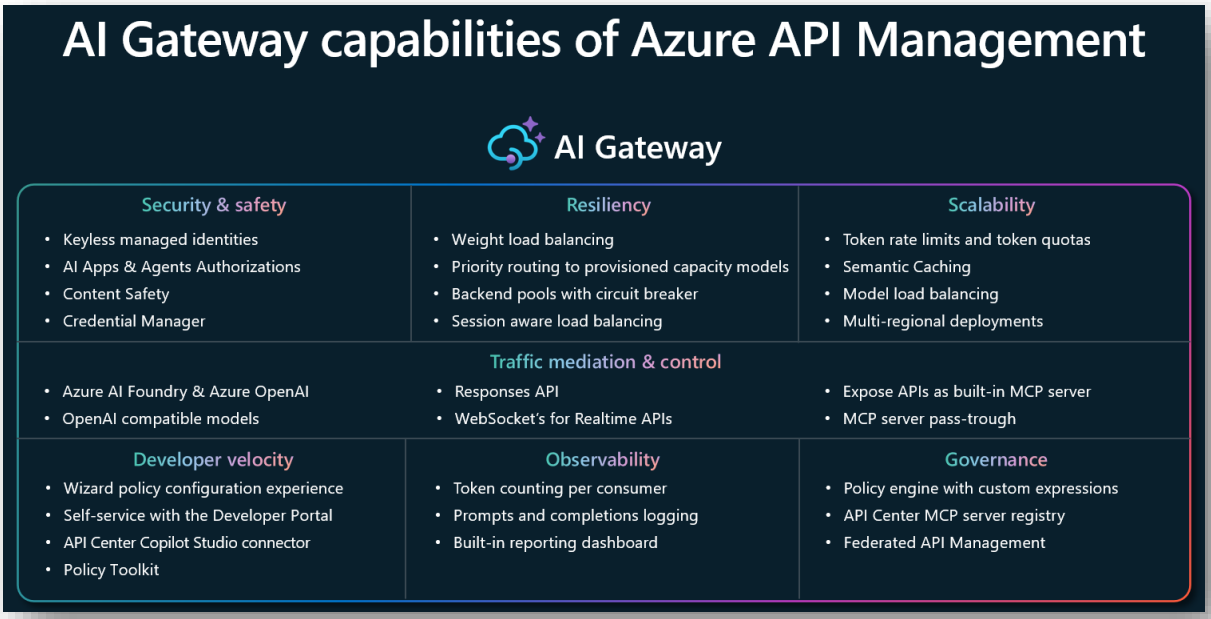
The AI release channel strategy aka.ms/apimdocs/updategroups, enables organizations to strike a balance between **reliability** and **velocity** by segmenting deployments into distinct stages—such as Canary, Preview, and General Availability.



Early-stage channels like Canary or Insider allow rapid iteration and testing of new features with a smaller, more controlled audience. This accelerates innovation and feedback loops while containing potential risks. As features mature and prove stable, they progress to broader release channels, ensuring that reliability is maintained for mission-critical environments. This tiered approach empowers teams to innovate quickly without compromising the stability expected by end users in production environments.

API Management updates are announced on the API Management GitHub repo. Subscribe to receive notifications from this repository to know when update rollouts begin.

This architecture yields an **enterprise AI gateway** with capabilities broken down into several key areas (a–g) described below.



a. Traffic Mediation & Control

One of the gateway's primary roles is **mediating traffic** between AI consumers and AI providers. Azure API Management serves as a **universal facade** that can front multiple models, data sources, and AI tools, abstracting their differences. It supports various API styles and protocols needed for modern AI workloads:

- **REST and Streaming APIs:** Most AI model endpoints (e.g. OpenAI's chat completions, Microsoft Foundry's inference API) use REST/HTTP. APIM natively proxies these, and importantly it also supports **streaming responses** (such as server-sent events) to enable enhanced user experiences. APIM also supports **WebSocket-based** AI APIs such as Azure OpenAI's real-time GPT-4o streaming for speech and audio. Developers can thus receive low-latency streaming outputs through the gateway. This means even voice or real-time chat interfaces can be mediated by the gateway, which passes through streaming data while still applying policies.
- **Multi-Model Routing:** The gateway can front multiple model backends (across Azure OpenAI, Microsoft Foundry, or even third-party AI providers) and route requests appropriately. For example, an enterprise could expose a single "ChatGPT" API that internally routes to different model providers or

deployments based on parameters or load. APIM's flexible **backend routing rules** can direct traffic to the correct model endpoint or to different versions of a model. This also enables **multi-cloud AI** usage through one gateway – e.g., Azure APIM could route some requests to an AWS Bedrock model and others to an Azure OpenAI model, while still applying uniform policies. In practice, APIM's backend entity supports round-robin, weighted, and priority-based balancing across multiple targets, allowing advanced routing strategies.

- **Protocol Mediation:** Beyond simple HTTP proxying, the AI Gateway can translate between client calls and backend protocols if needed. For instance, Azure API Management's support for the **Model Context Protocol (MCP)** allows it to act as a bridge between AI agents and tools (see Chapter 5). An AI agent might speak MCP (a JSON-RPC over HTTP+SSE protocol) to call a tool; APIM can expose an existing REST API as an MCP-compatible endpoint, effectively **mediating between JSON-RPC and REST** under the hood. This lets enterprise developers offer standardized "MCP tools" without rewriting their APIs – the gateway "speaks" the protocols on each side. In short, the gateway supports not just standard REST, but also emerging AI-specific protocols, making diverse tools and data accessible to AI in a controlled way.
- **Flexible Request/Response Handling:** APIM can perform request transformations and response filtering in transit. For AI scenarios, this might include injecting an organizational *system prompt* prefix on all incoming prompts or stripping out sensitive data from model outputs before returning them to the client. Such mediation ensures the AI interacts within allowed bounds – for example, adding a standard "company policy" instruction to every prompt invisibly. It could also mask or redact certain output content (e.g., replace detected Social Security numbers with [REDACTED]). These transformations are done via APIM policies (which can use code or regex), enabling fine-grained control over the content flowing in and out of models.

> In summary, the AI Gateway is the traffic cop that **connects various AI producers and consumers**. It abstracts the complexity of multiple model endpoints and protocols, presenting a unified interface to developers. By supporting WebSocket's for real-time interactions and MCP for agent-tool integrations, it ensures that as AI adoption grows (from simple REST calls to complex agent communications), the **enterprise has a single gateway to manage it all**.

b. Security & Safety

Security is paramount in an enterprise AI Gateway. Azure API Management provides multiple layers of security (🔗 Building a comprehensive API security strategy - eBook) to achieve a **zero-trust approach** for AI consumption, and it integrates safety mechanisms to ensure responsible AI use:

- **Managed Identities – Keyless Access:** Instead of embedding API keys for models in application code, APIM uses Azure **Managed Identities** to secure backend access. When APIM imports an Azure OpenAI or Foundry endpoint, it can configure a system-assigned identity that has permission to call that service. Clients then call the gateway **without needing any model key**, often using standard OAuth tokens or subscription keys for the gateway itself. This keyless approach means **no secret keys are exposed in client apps** – a huge security win. All calls to the AI model happen under the gateway's credentials, which are stored securely in Azure. If API keys are used (for third-party services), APIM can store them in Azure Key Vault 🔗 or its built-in credential store and never expose them to the caller.
- **Authentication & Authorization for Clients:** The gateway ensures every incoming request is authenticated and authorized. For internal use, a common pattern is to require callers to present an **Azure AD (Entra ID) OAuth2 token or a trusted client certificate**. APIM's 🔗 validate-jwt policy can enforce that the token is valid and belongs to an allowed user/app role. This means only approved applications or users can invoke certain AI APIs. Granular rules are possible: e.g., an "HR chatbot" API might only accept tokens from the HR department's app. APIM also supports its own subscription keys and IP restrictions for additional security layers. By centralizing authN/Z at the gateway, organizations avoid spreading credentials in each app and can update access rules in one place.
- **Credential Manager for Outgoing Calls:** When the AI Gateway needs to call external tools or APIs on behalf of an AI agent (see Chapter 5), it leverages a **credential manager** 🔗 to securely handle secrets. Azure API Management can maintain credentials (API keys, OAuth client secrets) for third-party services so that AI agents can call those tools via the gateway without directly handling any secrets. For example, if an agent needs to use a SaaS CRM API, the gateway can retrieve a stored OAuth token and attach it to the request. This prevents agents (and their underlying LLMs) from ever seeing raw credentials, and it centralizes secret management and rotation.

- **Content Safety Integration:** The gateway actively monitors and filters content to enforce responsible AI use. Azure API Management includes a built-in **content safety policy (** <llm-content-safety>**)** that can automatically moderate all incoming requests to an LLM API by sending them first to the Azure AI Content Safety service. If a user prompt contains hate speech, violence, sexual content, or other disallowed categories above predefined thresholds, the gateway can block or sanitize it *before* it reaches the model. This prevents malicious or inappropriate prompts from ever generating an output. Similarly, the policy could be configured to scan outgoing model responses and log or block those that contain unsafe content. By integrating Microsoft's content moderation at the gateway, enterprises put an automatic **AI guardrail** on every interaction. (The content safety policy can be tuned to specific categories and thresholds based on the organization's standards.)
- **Encryption and Network Isolation:** Like any enterprise system, APIM ensures data encryption in transit (HTTPS for all calls). In sensitive scenarios, the APIM gateway and Foundry can be placed inside a **private virtual network** with no exposure to the public internet. Client applications would access the gateway via secure VNET's or through an approved ingress (like Azure Front Door or Application Gateway with firewall). This way, even the model endpoints and gateway endpoint are not publicly accessible – the gateway becomes the sole controlled entry point. Additionally, APIM can integrate with Azure Web Application Firewall (WAF via Azure Application Gateway or Front Door) to provide protection against common web attacks (SQL injection, etc.), adding another layer of defense in front of AI APIs.
- **OAuth on Behalf of Users:** In advanced cases like AI agents acting for a user, the gateway can incorporate user-specific authorization. For example, if an AI agent needs to access a user's Microsoft 365 data via Graph API, APIM can perform an OAuth 2.0 **on-behalf-of** flow: the user authenticates once, and the gateway trades the user's token for a delegated Graph API token to call the tool. This way, the agent (through the gateway) calls Graph *with the user's privileges* and no more. The gateway ensures tools are invoked within the permitted scopes of the actual user, preventing an agent from over-reaching data it shouldn't have. This pattern keeps user-by-user data access under control without embedding user tokens in the agent (the gateway handles token exchange securely).

All these measures contribute to a **zero-trust, secure-by-design** architecture for AI usage. Every request is authenticated, **least-privilege access** is enforced (no uncontrolled keys or broad privileges), and content is inspected for safety. The AI Gateway therefore dramatically reduces security risks compared to a "wild west" direct model usage approach. *It lets enterprises reap AI's benefits without sacrificing control over security and safety*.

c. Resiliency

Enterprise-grade reliability is another hallmark of the AI Gateway. AI services can be unpredictable – rate limits may be hit, models might become unavailable or slow, or new versions roll out. The gateway mitigates these issues with resiliency features:

- **Load Balancing & Redundancy:** Azure API Management can define **backend pools** with multiple backend endpoints for an API. For AI models, this means you could have two or more deployments of a model (say in different regions, or one using reserved capacity and one using pay-as-you-go) serving the same logical API. The gateway will distribute traffic across them. APIM supports **round-robin, weighted, and priority-based load balancing**. For example, you can assign higher priority to a provisioned capacity instance (which has a fixed cost but guaranteed throughput) and lower priority to an on-demand instance so that the latter is only used when needed (overflow). This ensures optimal use of reserved capacity while still providing a fallback. If one backend becomes slow or fails, traffic can be shifted to the others seamlessly. Effectively, the gateway **virtualizes multiple model instances as one service** for the caller, handling the complexity of distribution and redundancy behind the scenes.
- **Circuit Breakers:** The gateway can detect when a backend is failing or overloaded and **stop sending requests to it** temporarily – implementing the *circuit breaker* pattern. In APIM, you can configure a backend with a trip policy (for instance, if 5 consecutive calls to that backend fail or timeout, mark it as "down"). Crucially, APIM's circuit breaker honors backend signals like Retry-After headers to know how long to wait before retrying that backend. For example, if an Azure OpenAI service responds "Retry-After: 10 seconds" due to throttling, APIM will pause forwarding traffic to that instance for 10 seconds. This prevents wasting requests on an unresponsive backend and allows it time to recover. Circuit breakers ensure **fast failover** – if one AI endpoint is not healthy, the gateway quickly routes around it to maintain overall service responsiveness.
- **Session-Aware Routing:** Many AI applications (especially chatbots or agents) require stateful interactions – ensuring a series of messages from one user go to

the same backend to preserve context. The AI Gateway offers **session-aware load balancing**: it can use a session cookie or custom session ID to consistently route an identified session to the *same* backend instance. This is crucial for multi-turn conversations where a specific model instance might be holding context in memory (for example, systems using Azure OpenAI's stateful **Assistants API**, **Responses API** or similar). With session affinity enabled, one user's chat won't be split between two model deployments mid-conversation. This feature ensures that scaling out models doesn't break the continuity of a dialogue. (If an instance does go down mid-session, the gateway could initiate a session transfer or notify the client to restart the conversation, but the goal is to minimize such disruptions.)

- **Failover and Multi-Region Deployments:** Enterprises often require geo-redundancy. You can deploy API Management gateways in multiple Azure regions (handled automatically with the premium tier) so that the AI Gateway spans regions. Users are then routed to the nearest region's gateway; if an entire region's gateway or services fail, traffic can automatically **fail over** to another region. Similarly, model deployments themselves (in Foundry or Azure OpenAI) can be deployed in multiple regions behind the gateway. Using a global traffic manager (like Azure Front Door or Traffic Manager) in front of APIM can direct clients to an available region. All of these ensure that even if a data center experiences an outage, the AI services remain available via another site. The gateway abstracts the multi-region complexity from the client – they just hit the global endpoint.
- **Throttling and Backpressure:** Resiliency isn't only about backend issues; it's also about protecting those backends from being overwhelmed. The AI Gateway's **rate limiting** policies (discussed more under Scalability) also serve to throttle traffic when needed. If an unexpected surge of requests comes in beyond what the models can handle, the gateway can reject or queue excess calls, rather than letting the stampede hit the model and potentially crash it. This acts as a form of backpressure to keep the overall system stable under stress. For example, if each model instance can safely handle 50 RPS (requests per second) and the gateway suddenly sees 100 RPS, it can enforce a 50 RPS cap per instance and send "too busy, try later" responses for the rest. This way, clients get quick errors rather than long timeouts, and the backend isn't brought down by overload.

In essence, the AI Gateway ensures that AI services are **robust and reliable** enough for enterprise SLAs. Even if individual model endpoints have limits or occasional failures, the gateway orchestrates them to meet higher availability requirements. Users of the AI services experience a smooth, continuous service – thanks to load balancing, smart routing, and fallback mechanisms working behind the scenes.

d. Scalability

Hand-in-hand with resiliency is **scalability** – the ability to handle growing load efficiently. The AI Gateway provides mechanisms to scale the usage of AI models in a controlled and cost-effective manner:

- **Token-Based Rate Limits and Quotas:** Traditional API gateways often rate-limit by requests-per-second, but for AI models a more natural unit is **tokens** (since one request might consume 10 tokens or 1000 tokens). Azure API Management introduced special policies to manage usage by token count. For example, the gateway can enforce that a particular app (identified by its subscription key or auth token) can use at most 500 tokens per minute and 50,000 tokens per day. If that limit is reached, further requests are immediately rejected or delayed until the quota resets. These **token quotas** can be set over various periods (hourly, daily, monthly, etc.) and are tracked centrally. This effectively **caps costs** because token usage correlates to billing – it ensures no single team or user can wildly exceed their budgeted usage. By enforcing token quotas, the gateway turns the AI model into a shared resource with controlled allocation, allowing many users to share it fairly. (For non-token-based AI APIs, APIM also supports traditional call-rate limits, but token limits are ideal for large language model usage.)
- **Scaling Out with Multiple Backends:** As described under resiliency, having multiple model instances behind the gateway also enables scaling out. If demand increases, the system can deploy another instance of the model (for example, spin up another Azure OpenAI deployment) and add it to APIM's backend pool. The gateway will begin by routing a share of traffic to it, effectively increasing throughput capacity. APIM's support for *weighted* distribution even allows you to ramp up usage gradually on the new instance (e.g., start at 10% weight, then 50%, then 100%) to ensure it is performing well. This horizontal scaling is transparent to consumers – they still call the same API – making it seamless to scale the serving layer without requiring any changes in client applications.

- **Semantic Caching:** A unique form of caching applicable to AI models is **semantic caching** of responses. APIM's AI policies include semantic cache store/lookup, which uses an embeddings-based mechanism to cache AI responses for *semantically similar* prompts. For example, if multiple users ask a question that is not exactly identical but essentially the same (e.g., "What is our vacation policy?" vs "Could you explain the holiday policy?"), the gateway can recognize the similarity via vector embeddings and reuse a cached answer instead of hitting the model each time. This can drastically reduce token consumption and latency for repeated or common queries. Under the hood, APIM integrates with a vector store (Azure Cache for Redis or the preferred Azure Managed Redis) to store embeddings and completions. The semantic cache policies generate an embedding for each prompt (using an embedding model like Azure OpenAI's text-embedding-3-small) and check for a semantic cache hit. If a match is found within a certain cosine similarity threshold, the stored completion is returned immediately. This not only speeds up responses but also **saves costs**, especially if the model is expensive, by avoiding redundant processing. It's an opt-in optimization that enterprises can enable scenarios like FAQ bots or knowledge base queries where questions repeat with slight variations.
- **Auto-scaling the Gateway:** Azure API Management itself can scale out to handle more requests. In the Premium tier, APIM allows scaling the number of gateway units (instances) and can be deployed across availability zones or multiple regions for higher capacity. For example, if the number of requests per second doubles, you might increase APIM from 2 units to 4 units to handle the load. The key point is the gateway can grow to handle more throughput as AI adoption increases, **without needing to redesign the integration** – you simply allocate more resources or upgrade the tier. This ensures that the governance layer itself does not become a bottleneck as usage expands.
- **Multi-Regional Deployments for Scale:** We touched on multi-region for resiliency, but it's also a scalability measure – distributing traffic across geographic regions brings the service closer to users (reducing latency) and spreads the load. An AI gateway can be part of a global architecture where requests are served in-region as much as possible, improving performance and concurrently increasing total capacity by leveraging infrastructure in each region. For instance, a gateway in the US and one in Europe can each handle local traffic, so the system's total capacity is the sum of both. Many enterprises adopt a "hub-and-spoke" or multi-hub model in their landing zone: e.g., a primary gateway in a central region and secondary ones elsewhere for regional offices. APIM handles configuration sync, so all gateways enforce the same policies, just operating in parallel.

- **Efficiency and Cost Optimization:** Scalability isn't just about raw capacity, it's about doing more with the same resources. The gateway's ability to **precompute prompt token counts** (and potentially reject overly long prompts before sending to the model) helps avoid wasteful calls. Its metrics allow identifying heavy users or inefficient prompts which can then be optimized (by developers or via caching). In FinOps terms (financial ops), the gateway is a tool to ensure scaling does not lead to runaway cost: by monitoring token use, enabling caching, and shifting load to cheaper resources, when possible (e.g., using a Small Language Model for simple requests or using reserved capacity first), the enterprise can manage the cost curve of scaling. We will see in Chapter 3 how FinOps considerations are baked in.

> Overall, these features mean the AI Gateway can **gracefully expand** to accommodate more usage while keeping performance high and **costs in check**. The enterprise can encourage more teams and users to leverage AI, confident that the gateway will throttle or scale as needed to maintain service quality for all consumers.

e. Developer Velocity

A perhaps underappreciated aspect of the AI Gateway is how it accelerates the **developer experience** in adopting AI. By providing an organized, self-service layer, it lets developers focus on building intelligent apps instead of worrying about integration plumbing:

- **One-Click Onboarding of Models:** Azure API Management offers a wizard to **import AI services**. Developers (or platform engineers) can simply point APIM to an Azure OpenAI Service resource or an Microsoft Foundry instance and select which deployed model endpoints to import. The APIM portal then automatically creates the API definitions, sets up the backend connection with managed identity auth, and even allows pre-configuring key policies (like token limits, logging, content safety, semantic caching, etc.) with simple checkboxes. This "few clicks" experience drastically simplifies exposing a new model through the gateway. For example, if a team fine-tunes a custom model in Foundry and deploys it, they can publish it via the gateway in minutes without writing custom proxy code or authentication wrappers – APIM's interface handles it. The time from *"model ready" to "model available as an enterprise API"* goes down significantly.
- **Self-Service via Developer Portal:** Once APIs (for models or tools) are published on API Management, developers across the enterprise can discover

and try them using the built-in **Developer Portal**. The Developer Portal is a user-friendly managed web app that lists available APIs, documentation, and lets developers test calls in-browser. For the AI Gateway, this means developers (mainly external) can find, say, a "Document Summary API" or a "Customer Support Agent API" exposed by the gateway, read how to use it, obtain credentials (subscription keys or get added to an Azure AD app role), and start coding against it quickly.

- **Azure API Center Integration:** Microsoft's **Azure API Center** serves as an enterprise API catalog. It can aggregate and display APIs from multiple APIM instances or other sources in one place. For AI Gateway usage, API Center can list all the AI-related APIs and even **MCP Tools** that the organization has published. This effectively becomes a "catalog of APIs and tools" available in the company. Crucially, API Center supports additional governance on these entries – metadata like the owner, description, usage guidelines, and even a workflow for publishing new APIs. It connects with developer experiences like **Copilot Studio** (for building copilots/agents) so that the AI APIs can appear as readily consumable connectors in low-code environments. The upshot is improved **discoverability**: whether a developer is working in Python or a business analyst is configuring a Power Platform chatbot, they all can easily find the enterprise-approved AI services in one place. This reduces the friction to use existing AI building blocks. This catalog model encourages re-use of AI capabilities and avoids duplicate efforts. If a language translation capability is already provided, another team can reuse it rather than calling a separate service or making a new integration.
- **Copilot Assistance for Policy Authoring:** Managing gateway policies can involve writing XML or configuration code, which might be complex for newcomers. However, Microsoft has introduced AI-assisted development here as well. Within the Azure Portal, there is an integration with the **Microsoft Copilot in Azure**. A developer can describe in natural language what policy they want ("limit this endpoint to 1000 tokens/day per user" or "mask any output that looks like a phone number") and the assistant will suggest the appropriate APIM policy snippet. Also, VS Code extensions for APIM provide integration with GitHub Copilot, IntelliSense and debugging for policies. These tools accelerate the gateway configuration tasks, making it easier to implement governance rules without deep expertise. **Policy samples and templates** aka.ms/apim/samples are also provided. The AI Gateway open-source repo

(Chapter 6) further offers ready-made policy sets. All of this reduces the learning curve and speeds up applying the gateway's features.

- **DevOps and Workspaces:** (We will discuss operational aspects more in Chapter 8) Briefly, APIM's **Workspace** feature enables multiple teams to work on the same gateway in isolation – each team can have their own workspace to manage *their* APIs and policies, which then roll up to the main instance. This federated model prevents the gateway from becoming a development bottleneck. Teams can independently iterate on their micro-APIs or AI endpoints without stepping on each other, while a platform team oversees global policies. Coupled with infrastructure-as-code (ARM/Bicep/Terraform) or the APIOps for CI/CD and source control, to propagate changes with proper review and automated deployment. This means developers can experiment quickly in dev environments, then promote their APIs to test and prod via workflows/pipelines, ensuring agility *and* compliance.
- **Familiar API Integration:** From a developer's perspective using an AI API via the gateway is the same as calling any REST API. They don't need special SDKs or to learn new protocols – the gateway presents a standard HTTPS endpoint with JSON payloads. This lowers the barrier to entry: any developer who can do an HTTP POST can integrate with a GPT-5 model or an AI-powered function call, without worrying about how to handle auth or logging (the gateway does it). APIM can even auto-generate example code or client SDKs for each API (the dev portal often provides "try it" code snippets in languages like C#, Python, JavaScript). This consistency speeds up development since teams don't have to adapt to different conventions for each AI provider. Popular AI SDKs like LangChain, LlamaIndex, and Semantic Kernel integrate seamlessly by simply updating the endpoint to the APIM gateway URL and using the APIM subscription key as the API key.

> In summary, the AI Gateway not only enforces rules but also serves as an **enablement platform** for development. It abstracts complexity and provides discovery, documentation, and tools that speed up the integration of AI into apps. By removing friction (security, scaling, etc. handled for them), developers can deliver AI-powered features faster and with more confidence. In practice, companies have found that having a centralized gateway and catalog for AI services allows them to roll out new AI-driven capabilities in days rather than weeks, because much of the boilerplate (obtaining keys, adding logging, setting up monitoring) is already solved uniformly.

f. Observability

The AI Gateway embodies this principle by offering deep **observability** into AI usage. It provides rich telemetry about every request, which is crucial for governance, troubleshooting, and continuous improvement of AI applications:

- **Token Counting per Consumer:** The gateway tracks detailed metrics on token usage for each request and for each consumer (app/team). APIM's policies can emit custom metrics to Azure Monitor (Application Insights) that include fields like *prompt tokens*, *completion tokens*, *total tokens* used by the request, as well as identifiers like which API and which subscription/user made the call. This allows creation of dashboards on how many tokens each team or application is using over time – vital for cost tracking and optimization (FinOps). Essentially, the gateway turns the abstract concept of "AI usage" into concrete data. For example, a dashboard might show that the Finance chatbot consumed 1.2M tokens this week, whereas the R\&D agent consumed 300k, which helps in cross-charging cloud costs or identifying unexpected usage spikes.
- **Prompts and Completions Logging:** The gateway can log the **content** of prompts and responses (with controls to avoid storing sensitive info if needed) to an audit log. By enabling Azure Monitor diagnostic settings on the APIM instance, *every* LLM request/response can be recorded, including the model's name used and the full text of the prompt and completion. Large prompts or outputs are split into multiple log entries that are sequentially numbered for aggregation. This is incredibly useful for debugging (seeing exactly what prompt led to a weird model response), **audit trails** (reviewing AI interactions for compliance or ethical issues), and **prompt engineering improvements**. For example, reviewing logs may reveal that users often phrase a query a certain way that the model handles poorly – developers can then adjust the prompt or provide better instructions. Having a centralized log also means if a concern arises (say, "did our bot ever output inappropriate content to a user last month?"), the logs can be searched to find that out – something near impossible without a gateway.
- **Built-in Reporting Dashboard:** Microsoft has provided a built-in "AI usage" dashboard experience in the Azure Portal for APIM's AI gateway features. This dashboard gives at-a-glance visualizations of token usage trends, top APIs by consumption, error rates, and content safety incidents caught, etc. For example, it might show a graph of total tokens by day with a breakdown per API or per department, highlighting if usage is trending or if any team exceeded their expected usage. These insights make it easy for stakeholders (even non-

developers like project managers or FinOps analysts) to monitor AI use across the organization. Since it's in Azure, one can also customize and extend these reports using Log Analytics Workbooks or Power BI to create executive summaries such as "AI Cost per Business Unit vs. Value Delivered."

- **Monitoring and Alerts:** Since all metrics and logs flow into Azure Monitor, organizations can set up **alerts** on important conditions. For instance, an alert could trigger if a particular app's token usage today is 50% higher than the same time yesterday (could indicate a bug or runaway script). Or an alert if **any** request is blocked by content safety for "critical" severity content (so compliance officers are notified immediately of a policy violation attempt). Another might watch the success rate of the model API – if error percentage goes above 5% in a given hour, alert the on-call engineer. By having a single funnel for requests, it becomes much simpler to monitor the system health and usage from one place, rather than aggregating logs from many apps. This real-time awareness is key to maintaining reliable operations and quickly addressing issues.
- **End-to-End Traceability:** APIM assigns request IDs and can integrate with distributed tracing systems (e.g., Application Insights correlation IDs). This means a particular user action can be traced across the system. For example, if a user says "order me more paper" to an AI assistant, you could trace: the request ID from the client, through the gateway (which logs the prompt), to perhaps an internal API call or MCP tool that the agent triggered. In agent scenarios with multiple tool calls, the gateway logs each tool invocation. This provides a **trace of multi-step agent actions**, which is invaluable to debug complex agent behaviors. If an agent made 5 tool calls and something went wrong on the 5th, you can see the sequence and inputs to each, which helps in diagnosing logic issues or errors.
- **Cost and Performance Analytics:** Observability data can feed into business metrics. By logging token usage along with subscription Id, model IDs and operation, companies can calculate the cost incurred by each model or feature. For example, "Feature X used 100k tokens on GPT-4 this week, costing ~$200, whereas Feature Y used 100k on GPT-4-1-mini, costing ~$20." This kind of analysis can guide decisions like whether a cheaper model is sufficient for a given task. Performance metrics (latency, throughput per model) can also be compared – e.g., if a new model version is slower, it will show in the logs. In essence, the gateway's data allows enterprises to assess **"How is our AI being used and is it delivering value proportionate to its cost?"** and get quantitative answers. It brings transparency to AI ROI.

- **Auditing and Compliance Logs:** Having a central record of AI usage helps with compliance requirements. If regulators or internal auditors ask for records of AI system outputs, the logs are available. The organization can prove that, for example, content moderation was applied (by showing log entries of blocked prompts) as part of Responsible AI practices. It can also help with data governance: if a user requests deletion of their data, you can search the logs to ensure any prompts containing that data are handled appropriately. In fields like finance or healthcare, where record-keeping is mandated, such logging is essential if AI systems are involved in customer interactions.

> With these observability features, the AI Gateway turns what could be a "black box" interaction with an external AI service into a **transparent, monitorable process**. Stakeholders from IT, to finance, to compliance can get the insight they need, whether it's usage stats, cost metrics, or content logs, all from the gateway's data. This feedback loop is crucial for continuous improvement (as we'll discuss in Chapter 8). For example, one organization used token logs to identify inefficient prompts that were driving up costs and then optimized those prompts to save money – a direct outcome of measuring and observing the AI usage.

g. Governance

Governance is a broad theme that spans many of the above points, but to highlight specific governance structures the AI Gateway enables:

- **Policy Engine & Custom Rules:** Azure APIM's policy engine is extremely flexible (policies are configured via XML and allow C# expressions). This allows creation of **custom governance rules** tailored to enterprise needs. For example, a company could require that every AI request include a header or claim identifying the business unit and write a policy to reject calls missing it – ensuring traceability and proper chargeback. Or a custom expression can route models to the most suitable endpoint or region where they are available. Essentially, if the out-of-box policies don't cover a scenario, an organization can often script a solution in the pipeline. This is far easier than trying to implement checks in each application. By encoding business-specific requirements at the gateway, you ensure **uniform enforcement** across all uses of AI. This includes ethical guidelines too – for instance, if corporate policy forbids using a non-vetted model, the gateway could block calls to any model ID that isn't on an approved list.
- **Cataloging and Lifecycle Management:** Governance also means knowing what AI assets exist and controlling their lifecycle. By using Azure API Center as a

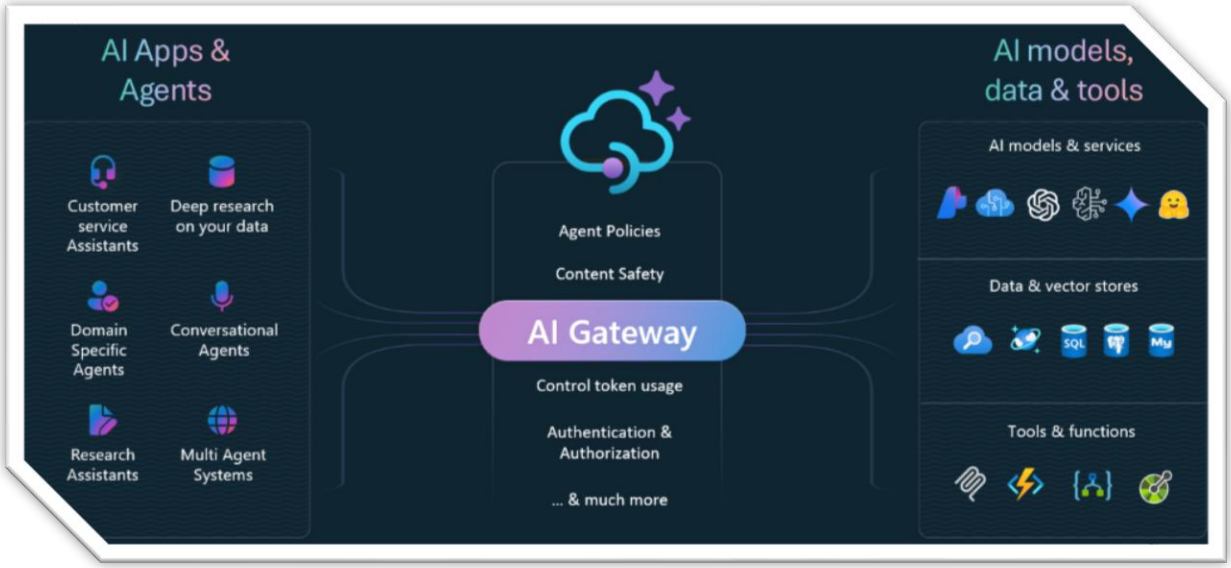
catalog (and registry for MCP tools), the enterprise gains a bird's-eye view of all published model APIs and tools. Every API is a managed asset with an owner and description. Nothing gets deployed without being registered. This prevents "shadow AI" uses – i.e., an enthusiastic developer exposing a new AI model without oversight. Through the catalog, governance teams can **approve new APIs/tools** before they go live, ensuring things like a compliance review or security testing has been done. The gateway aids version management too: if a model is updated to v2, you can publish it as a v2 API while still keeping v1 for a deprecation period and monitor the usage until v1 is phased out. APIM allows running multiple versions of an API simultaneously and even adding deprecation warnings.

- **Federated API Management:** Large enterprises may have multiple teams managing subsets of APIs. The AI Gateway supports a **federated model** through APIM Workspaces and role-based access control. Each team or business unit can be delegated rights to manage their own AI APIs and policies within the gateway, while a central platform team oversees global settings and policies. This federated approach balances agility and control: teams have freedom to iterate on their AI services, but within guardrails defined by central IT (which might mandate certain baseline policies). It also means the gateway can scale organizationally – it's not one group becoming a bottleneck for every change. Essentially, governance responsibilities are shared: local teams govern the details of their service, central teams govern the framework and standards.
- **Integration with Data Governance (Purview):** As discussed in Chapter 4, the gateway can integrate with broader data governance tools like Microsoft Purview. For governance, this means the policies defined in Purview (e.g., sensitivity labels, data access rules) can be enforced at the gateway. If Purview classifies a certain API as containing sensitive data, the gateway could automatically require extra authentication steps or block external use of that API. Also, by funneling data access through the gateway, Purview's lineage tracking can capture how data was used in AI outputs, which is important for compliance (e.g., if asked "did any AI output use data from Database X?" one could trace via gateway logs). The AI Gateway effectively becomes the **technical enforcement point** for data usage policies in AI scenarios.
- **Compliance and Risk Policies:** If industry regulations require certain AI interactions to be restricted or recorded, the gateway is where to implement it. For example, GDPR might require that personal data isn't sent to certain AI services without consent. A gateway policy could detect if an API call contains

what looks like personal data (via integration with Content Safety blocklists, a data classification service or regex patterns for things like emails, phone numbers, credit cards, social security number) and block or anonymize it unless a consent flag is present. Another example: a bank might have a rule that any AI model used for customer communications must have undergone bias testing. The gateway could enforce that only specific model IDs (those that passed review) are accessible and reject calls to any other model. Essentially, high-level AI governance decisions (often documented in Responsible AI guidelines) can be translated into programmatic rules at the gateway.

- **Enterprise Standards and Best Practices:** The gateway helps institutionalize best practices. Suppose there is a known risk like prompt injection attacks – the security team might develop a mitigative prompt prefix to reduce that risk. Through the gateway, they could automatically prepend that prefix to all user prompts for certain models, enforcing the best practice enterprise-wide. Or if it's discovered that a certain output format is needed for downstream systems, the gateway could ensure all responses conform by transforming them or adding fields. By capturing these patterns once in the gateway, all teams benefit; they don't each need to re-learn or implement the fix. This raises the baseline of quality and compliance for all AI applications in the company.

> In summary, **governance via the AI Gateway** means that as AI proliferates in the enterprise, it does so in a **controlled, compliant, and cost-effective manner**. The gateway acts as the centralized nervous system, where policies and oversight are applied uniformly. This gives management confidence that even as dozens of AI projects run in parallel, they all adhere to the organization's standards for security, quality, and ethics. It turns governance from a manual, after-the-fact process into an automated, in-line part of the AI pipeline.



At this point we have covered how Azure API Management and Microsoft Foundry provide a powerful foundation for an AI Gateway, with capabilities (a) through (g) that span traffic control, security, resiliency, scalability, developer experience, observability, and governance. In the next chapters, we will drill down on specific areas: how the gateway governs AI *Models* (Chapter 3), *Data* (Chapter 4), and *Tools* (Chapter 5), followed by practical implementation and operational guidance (Chapters 6–9).

Chapter 3: Governing AI Models

With the foundation in place, let's focus on governance specifically for **AI Models** – i.e., how the AI Gateway governs the consumption of models (like GPT-5, open-source models and third party LLMs/SLMs) to ensure they are used safely and cost-effectively. This involves applying many of the capabilities from Chapter 2 in targeted ways.

Safe and Controlled Model Consumption

The gateway mediates every call to an AI model, which means it can enforce **who can access which models and how**. For example, an enterprise may have a highly powerful reasoning model deployment that only certain mission-critical applications should use. Through APIM's product/subscription model, or token authorization, the gateway ensures that only authorized applications with valid credentials can access the model's API. Even within an allowed app, the gateway might enforce that only certain operations or model parameters are permitted. This prevents misuse, such as a low-importance script accidentally calling the most expensive model when a cheaper one would do.

It also enforces **input validation policies** – e.g., if a model should only be fed non-sensitive data, a policy could detect and redact any occurrences of personal identifiable information (PII) in requests (perhaps by calling a data anonymization function 🔗 before forwarding to the model). All these measures govern the *safety* of model usage, making sure models are invoked in approved ways. In essence, the gateway acts as a *gatekeeper* per model: only authorized clients, sending appropriate content, can trigger a given model.

Another aspect is **model-specific routing**. If multiple models are available, the gateway governance can decide *which* model gets called for a given request. For instance, a policy might say: if the requested task is "summarization" and the document length is small, use the cheaper summarization model; if it's very large or critical, invoke the more advanced model. By doing so transparently at the gateway level, you ensure the right model is used for the right job, according to governance guidelines (this overlaps with cost optimization below).

Finally, safe consumption implies **monitoring outputs** for appropriateness. The content safety integration we discussed can apply per model as needed. If one particular model tends to give edgy outputs, the gateway could enforce an additional layer (like always run its output through a filter or an approval step) for that model's responses.

Cost Governance and FinOps for AI

Despite the general trend of decreasing costs, AI models—particularly large reasoning models—can still incur significant expenses. A core part of governing models is managing these costs across the organization, which is where **FinOps for AI** comes in. FinOps (Financial Operations) is the practice of optimizing cloud spend; applied to AI it involves tracking and controlling model usage costs. The AI Gateway is instrumental in FinOps for AI in several ways:

- **Visibility of Costs:** As discussed in Observability, the gateway provides a breakdown of token usage by application/team. FinOps teams use this to allocate costs to the correct budgets (show back or chargeback). For example, if the marketing department's chatbot used 100 million tokens at an average cost of $0.375 per million tokens, FinOps can attribute $37.5 of AI inference costs to the marketing budget. The gateway's logs provide detailed per-token and per-application cost metrics, even if the AI provider's raw billing is aggregated. This transparency raises departmental awareness of AI usage.
- **Setting Quotas and Budgets:** FinOps best practices suggest putting budgets or caps to prevent overspend. The gateway's **token quotas** are essentially budgets in technical form. A team might be given, say, 500k tokens per week as their "budget"; the gateway enforces that limit. If they hit the quota, further usage is halted or gets a "quota exceeded" error until it resets (or someone intervenes to raise it). This is akin to how cloud budgets or Azure subscriptions have spending limits, but it's much finer-grained and immediate. It turns a financial control into an engineering control. Managers can adjust quotas as needed, but the existence of a cap means no team can unknowingly run up a massive bill.
- **Preventing Runaway Costs:** Sometimes a bug or misconfiguration can cause an explosion in AI calls (for example, an infinite loop that calls a model repeatedly). Without a gateway, this could run unnoticed until a huge bill arrives. The gateway's combination of rate limits and quotas will throttle that. Also, FinOps anomaly detection can be applied to the metrics the gateway emits – e.g. setting up an alert if today's token usage for an app is double its 7-day average. By catching anomalies early (the gateway could even automatically email the owner of a subscription if usage spikes abnormally), it prevents costly surprises.

In effect, the gateway is the circuit breaker for cost as well — not just technical failures.

- **Optimizing Usage Patterns:** With detailed logs, organizations can analyze how models are used and find optimizations. Perhaps they find that 30% of tokens are being wasted on very long prompts that include irrelevant context. Developers can be trained (or required) to shorten prompts or use more efficient input structure. The gateway could even enforce a max prompt length or strip out parts of a prompt that it knows are excessive. Or logs might show certain queries repeat often, suggesting enabling semantic caching to save money (cache hits cost nothing extra). FinOps for AI encourages techniques like **prompt engineering to reduce tokens** and **caching** to avoid repetitive calls – both of which the gateway facilitates (prompt modifications via policy, caching via semantic cache). By iterating on these optimizations, the enterprise can significantly cut costs while delivering the same AI outcomes.
- **Choosing Cost-Effective Models:** The gateway also allows **A/B testing** different models behind the scenes. For example, one might route 90% of requests to Model A (expensive) and 10% to Model B (cheaper) and compare results (captured via logs or user feedback). If quality is comparable, more traffic can be shifted to the cheaper model, saving money. This kind of dynamic model selection can be orchestrated by the gateway (with weighted routing and analyzing outcomes in logs). Over time, the organization ensures each use case is using the **most cost-effective model that meets requirements**. FinOps is about balancing performance and cost – the gateway empowers those decisions by making model switching configurable and measurable.
- **Chargeback and show back:** Using the gateway's per-team metrics, a company can implement chargeback (billing departments for their usage) or at least show back (reporting their usage costs). This tends to drive accountability – when teams see a monthly report of "you used 100M tokens = $37.5", they're motivated to optimize their usage or justify it with business value. Without a gateway, such granular reporting is nearly impossible, especially if multiple teams share the same model endpoint or API key. The gateway *splits the usage* by consumer, making cost accountability feasible, which is a cornerstone of FinOps culture.

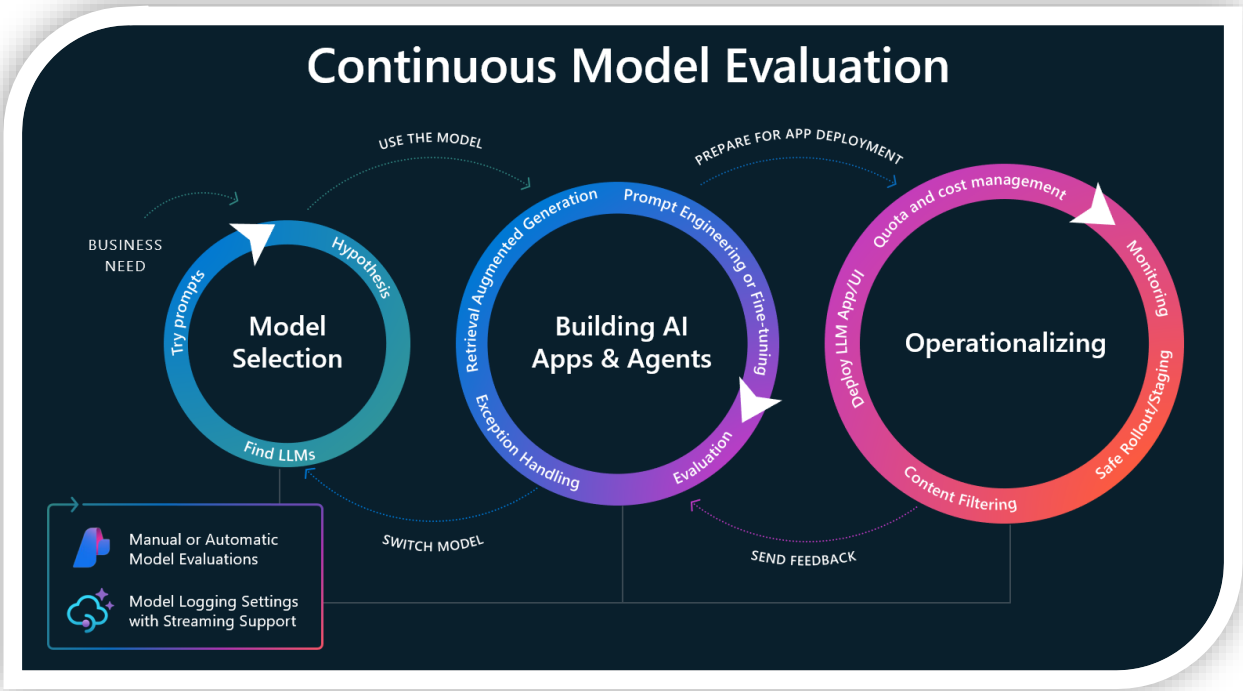
In the broader FinOps picture, **the gateway is the enforcement arm for cost policies**. As the FinOps for AI overview notes, real-time monitoring of metrics like tokens is critical for preventing budget overruns, and throttling can be used for cost control at peak times – exactly what the AI Gateway implements. It turns abstract best practices into concrete controls. Follow this lab 🔗 for a practical FinOps for AI implementation.

Ensuring Model Effectiveness and Quality

Governance of models isn't just about restricting usage and cost; it's also about making sure the models are delivering value and being used appropriately:

- **Monitoring Outcomes:** The gateway logs can feed model evaluation processes. By capturing a sample of prompts and completions, AI teams can perform offline analysis to see if the model responses are high quality or if another model would be better. Perhaps the completions from the current model are too verbose (costing many tokens) – a shorter, fine-tuned model might yield more concise answers, saving tokens and improving user experience. Without gateway logs, gathering such real-world usage data reliably is difficult. With them, teams can actually see how the model is being used in production and tune accordingly.
- **Continuous Model Improvement:** If multiple models are available (say GPT-5, GPT-4.1 and o3), the gateway can facilitate **comparative experiments**. We mentioned A/B testing for cost, but also for *quality*, one could intentionally route some traffic to each model and solicit user feedback or measure resolution rates. The gateway track the model used for each completion, to enable downstream analysis. Over time, this ensures the organization is always using the **"most performant model" for each task** – updating gateway routing to use better models as they become available. Essentially, the gateway makes model usage *configurable*, so switching out a model doesn't require app changes – it can be done centrally when confidence in a new model is established.
- **Ethical and Policy Compliance:** For certain models, governance might require special safeguards. For instance, using an open-source LLM that hasn't been rigorously tested might be limited to internal use only or behind additional filters. The gateway can enforce these boundaries (like only allowing that model's API to be called from certain IP ranges or by admin accounts). This ensures experimental or less-trusted models don't accidentally end up in customer-facing scenarios without oversight. Conversely, if a model is designated for sensitive content, the gateway could enforce that *only* that model is used for those scenarios (to ensure compliance). Basically, the gateway can standardize which models are allowed for which purposes, aligning with the organization's responsible AI policies.
- **FinOps Benchmarking:** FinOps for AI also involves measuring business value relative to cost. By governing model use through the gateway, the organization can start attaching outcomes to usage. For example, if a particular AI model call

correlates with handling a customer support query, one can derive cost per query handled. That can feed decision-making: is it worth using the more expensive model if it resolves issues 10% faster? These decisions are strategic, but they rely on the **data that governance provides**. The gateway is the source of truth for how models are used and at what cost; leadership can then make informed calls on AI investments (like deciding whether to invest in fine-tuning a cheaper model vs. paying for more of an expensive model's usage).



In summary, **governing AI models via the gateway ensures two things**: *safety* (the right people use the right models in the right way) and *efficiency* (monitoring and optimizing the cost/value of model usage). By applying FinOps principles with the granular controls of the gateway, enterprises turn AI from a wild new expense into a manageable, trackable utility. They gain the ability to answer: "How much are we spending on AI? Who is using it? Is it worth it, and how do we make it more efficient?" – which is a cornerstone of responsible, sustainable AI adoption.

Chapter 4: Data Governance for AI

AI models are only as useful as the **data and knowledge** we give them. In enterprises, a large part of AI's power comes from integrating internal data – databases, documents, analytics – into AI applications. This chapter covers how the AI Gateway helps govern access to **enterprise data** in AI scenarios, ensuring that data is used responsibly and effectively.

APIs as the Gateway to Enterprise Data

Modern enterprises typically expose data via **APIs** (RESTful services, etc.) rather than direct database connections. This is even more important for AI, where an LLM or agent can't/shouldn't directly connect to a database or file share without an intermediary. The AI Gateway concept extends to these data APIs as well. Essentially, any enterprise data source that an AI might need (customer records, knowledge base articles, inventory data, etc.) should be accessed through a managed API that the gateway can control. This achieves two things:

- **Security and Compliance for Data Access:** By funneling data access through APIs under gateway management, we can apply all the API Management security features (OAuth, role-based access, logging) to data retrieval. So, if an AI agent wants to fetch a customer's profile, it must call a "Customer Profile API" via the gateway, which can then check: is this agent (or the user it's acting for) allowed to access that customer's data? Should certain sensitive fields (like SSN) be redacted for this call? And all access is logged for audit (for example, those logs could be consumed by Purview or a SIEM to track data usage). This controlled access ensures **data governance policies extend into AI** – the AI agent isn't doing anything a normal app couldn't do, because it's using the same governed APIs.
- **Consistency and Monitoring:** It prevents scenarios where an AI agent might otherwise try to scrape data from some source or use a browser (aka computer use) in an untracked way. If everything the AI sees from enterprise systems is served via an API call that's governed, it dramatically lowers the chance of data

misuse or leakage. No call goes unnoticed or unregulated. It also means if something goes wrong (say an agent pulls a lot of data it shouldn't), it's visible and can be stopped quickly by revoking API access or adjusting the policy.

In practice, this means enterprises should create APIs for any data they plan to feed to AI, if they don't exist already. Many companies already have APIs for their services (microservice architecture); the gateway approach leverages that by simply treating those internal APIs as potential "tools" or data sources for AI (to be invoked via the gateway with all the same security).

Integration with Azure AI Search (Vector Store)

One popular pattern to inject enterprise knowledge into LLMs is **retrieval-augmented generation (RAG)**, where you provide the model with relevant documents or knowledge base articles from a search index. Azure AI Search, especially with its vector search capability, is often used to index enterprise content and retrieve relevant chunks given a user query embedding. The AI Gateway can integrate with this pattern in a couple of ways:

- **Search API Proxy:** The gateway can host a **search API** that AI agents call to get context. For example, an AI agent might use the Azure AI Search SDK using the APIM gateway URL as the search endpoint. The gateway then forwards this request to Azure AI Search and returns the search results to the agent. By doing this via the gateway, all those search queries are logged and governed. The gateway can apply policies like rate limiting (so an agent doesn't overload the search index with too many queries) or even content filters on the queries (to ensure an agent isn't searching for disallowed info). It also ensures **security trimming** is respected: AI search can be configured to respect user roles, but the gateway can add an extra layer by including the user's identity or permissions in the search query and ensuring the agent only gets results the user is allowed to see. In short, the gateway acts as a broker to the vector store, controlling how AI accesses unstructured data.
- **MCP Tool for Search:** If the agent uses the **Model Context Protocol** to retrieve data, APIM can expose Azure AI Search as an **MCP tool**. Essentially, APIM can present an endpoint that follows the MCP spec (like a JSON-RPC method *searchDocuments*) which internally calls AI Search. This way, even if the agent is using an advanced agentic framework that speaks MCP to all tools, the *actual implementation* of the search tool is still a governed API call under APIM. It's just packaged in a different protocol. The result is the same: the gateway mediates the AI's vector database access.

From a governance standpoint, by integrating AI search, we ensure that the **data being provided to the model is relevant and approved**. The AI isn't blindly reading a file share or database; it only sees what the search index returns. And that index can have built-in filters (like security trimming). So instead of an LLM scanning thousands of documents uncontrolled, it's getting, say, the top 3 relevant paragraphs *which have already passed through security filters*. This confines the model's knowledge to what it should know. If the agent tries to abuse search (like querying very broadly), those calls are still subject to monitoring and rules.

Integration with Azure Logic Apps

Azure Logic Apps offers over 1,400 Microsoft-managed 🔗 connectors for seamless, secure integration with various data sources such as SharePoint, Oracle DB, Salesforce, and SAP. These connectors provide a range of triggers and actions, allowing automated workflows to run on schedules or in response to specific events (e.g., document uploads to SharePoint). This flexibility enables users to build efficient document ingestion pipelines and knowledge bases, such as those utilizing vector embeddings in Azure AI Search.

Responsible AI – Content Safety on Data Outputs

We've already discussed content safety for model prompts and outputs. Another angle is ensuring that if an AI is allowed to output *enterprise data* to users, it doesn't reveal sensitive content inappropriately. The AI might have access to internal documents and could potentially include confidential info in its answer to a user. So, the same content moderation concepts apply, but even beyond the standard categories (hate, sexual, etc.), we worry about sensitive *enterprise* data (trade secrets, personal data).

The AI Gateway can implement checks such that if an AI's response contains certain sensitive terms or patterns, it could mask or remove them. This could be done via a custom policy or by integrating with data classification systems:

- **Data Classification Labels:** Microsoft Purview can classify and label data for sensitivity (e.g., "Highly Confidential"). If documents in the AI search index are labeled, an advanced setup might tag the content when retrieved. The gateway, upon seeing a piece of content with a certain label, could either prevent it from being shown to an end-user who isn't authorized or redact parts of it. This is not trivial out-of-box, but conceptually feasible. It might involve storing metadata with the search results and having a policy script in APIM that checks those labels against the user's clearance.
- **Regex and Patterns:** A simpler approach is to use APIM's ability to replace or mask content using regex. For instance, the gateway could scan outbound

responses for things that look like social security numbers, or internal project code names, and replace them with "*". This would be a catch-all that doesn't rely on labels. It's not foolproof but can help prevent obvious leaks (like if by chance an internal document with sensitive numbers was being echoed by the AI).

Azure Content Safety primarily targets harmful content, but Microsoft also has the Azure AI Language service that can detect and redact Personally Identifying Information in any text. The gateway can serve as a place to implement those *whenever data is leaving the system*. Essentially, treat any AI response as another thing that might need sanitization before going to an end-user.

The key point is that the gateway allows insertion of these controls without altering the model or the data source: it's a middleware approach to Responsible AI. If the AI accidentally puts a confidential code snippet in its answer, a well-crafted gateway policy could catch that and strip it out or flag it for review, thereby averting a data leak.

Integration with Microsoft Purview for Data Governance

Microsoft Purview is the unified data governance and catalog solution for enterprise data. It catalogs data sources, tracks lineage, and defines data usage policies. How does this interplay with the AI Gateway?

- **Data Catalog informs AI Tools:** Purview can be the inventory of all data assets and their sensitivity. The organization might decide which data sources are allowed to be used in AI scenarios and under what conditions. For example, Purview might mark a SQL database as containing customer financial data – policy might dictate that this data cannot be used in AI model prompts unless masked. The gateway can enforce those rules. One way: every API that surfaces that finance data could be tagged in APIM and classified in API Center as "sensitive", and the gateway has a global rule: "if an AI request tries to call a sensitive data API and the target model is an external service (like OpenAI), block it unless it's gone through an approval process." This kind of integration between data classification and runtime policy is emerging. Purview doesn't directly plug into APIM policies as of today, but one can imagine custom solutions where Purview metadata is exported to the gateway config (for example, injecting allowed/denied named values per data classification).
- **End-to-End Lineage:** Purview's lineage can show how data flows through systems. If we incorporate AI, we'd want lineage that a piece of data went from System A, through an API, into an AI model which then produced output B. This is challenging, but the gateway logs provide a linkage: they can tie a request for

data to a subsequent model call. If those logs or identifiers are fed back to Purview, one could at least document that “Dataset X was used in Prompt Y on date Z”. This is an area likely to evolve. For now, it might be more manual: e.g., analyzing logs and manually annotating lineage. But the gateway makes it possible by having the logs of those interactions.

- **Holistic Governance:** Purview’s aim is to ensure data is well-managed (classified, quality-checked, access-controlled). The AI Gateway’s role is to **implement** those access controls in real-time for AI. For example, Purview policy might say "Customer PII data must not be exposed to third-party systems." The gateway, knowing that a particular model API ultimately calls an external model (like third party hosted model), can enforce that any API call including PII to that model is blocked or anonymized. Without a gateway, an AI might send PII inadvertently; with the gateway, you have a choke point to enforce “no PII goes out” by hooking into content scanning or requiring that any PII fields are redacted (maybe by forcing the app to use a special endpoint that only returns masked data).

In summary, Purview defines the *rules* and *inventory* of data, and the AI Gateway is one of the enforcement points to ensure AI usage of data follows those rules. As Microsoft’s data governance and AI governance tools become more integrated, the gateway will serve as the central point of connection between them.

Integration with Data Services (Azure SQL, Fabric, etc.) trough MCP

Enterprises have numerous data services: Azure SQL DBs, Cosmos DB, Microsoft Fabric, Databricks, etc. Many are accessible via APIs or at least via SDKs. The **Model Context Protocol (MCP)** is emerging as a universal way for AI agents to use *tools*, including tools that query data services. Here’s how the gateway helps in those cases:

- **API Wrappers for Data Operations:** Suppose we want an AI agent that can answer “What were our sales last quarter?” The data is in a SQL database or a Fabric warehouse. Rather than give the agent direct DB connectivity (which is not secure or feasible), we create an API (or use an existing one) that runs the SQL query and returns the result (e.g., /getSalesData?quarter=Q4). We then expose that API as a *tool* via the gateway (maybe as salesDataMCP tool). So, the agent simply invokes the “GetSalesData” tool, and through the gateway it calls the underlying API with proper authentication and gets the result. Throughout this, APIM manages credentials (so the agent never sees DB passwords) and

logs the query, and ensures the call is authorized (the agent or user had permission to access sales data). This approach leverages API-ization of data to allow AI to use data safely.

- **MCP Registry in API Center:** As mentioned, API Center can act as a **registry for MCP servers/tools**. So, all these data-access tools can be listed there. Perhaps you categorize them under "Data Tools": e.g., “SalesDataTool”, “InventoryLookupTool”, each backed by some API. The gateway ensures if tomorrow the data source changes (we migrate the sales data from one DB to another), we just update the API backend implementation; the tool and its interface remain the same to the agent. This decoupling through MCP + Gateway means **data sources can evolve without breaking AI agents**, which is a big maintenance benefit.
- **Response Transformation:** Additionally, data responses can be transformed if needed. Maybe a raw SQL result returns a lot of columns, but we only want the AI to see a summary. APIM could have a policy that post-processes the API’s JSON to only include certain fields or aggregate the data. That’s data governance in action: giving AI only what it needs and nothing more.

In essence, **the AI Gateway becomes the broker for all data that flows to and from AI models/agents**, which is exactly what you want for proper data governance. It ensures:

- Data requests are *authenticated, authorized*, and tracked.
- Sensitive data is protected and only accessed in approved ways (the gateway won’t let an agent bypass security).
- There’s an audit trail of what data was used for what AI query (so later we can answer “how did the AI know X?”).
- The organization can leverage existing data governance investments (Purview, content safety) by connecting them to this centralized AI usage point.

> By integrating vector stores (for unstructured data retrieval), content safety or AI language services (for filtering output), Purview (for oversight), and treating internal data services as governed APIs, the AI Gateway ensures **enterprise data remains under control in the age of AI**. Users and AI agents get the benefit of AI having knowledge of company data (like an assistant that knows your internal knowledge base), but the company maintains confidence that this data isn’t misused or leaked, and that compliance requirements are met.

Chapter 5: Tools Governance for AI

Beyond models and data, **tools** (functions, external services, or in general actions that AI agents can invoke) are a crucial part of advanced AI applications. Generative AI agents often augment their capabilities by calling external tools – for example, an agent might call a weather API, send an email via an SMTP service, or execute a transaction in an ERP system via an API. Governing these *AI tools* is just as important as governing models and data. The AI Gateway plays a central role here, especially with the advent of the **Model Context Protocol (MCP)**, which is emerging as a standardized way for AI to interact with tools.

Why Tool Governance is Important

If an AI agent can call arbitrary tools or execute actions autonomously, there are risks: it could retrieve information from an untrusted source, perform an action it shouldn't (like making an unauthorized financial transaction), or leak data to an external API. Essentially, you need to **sandbox AI agents' tool use** similar to how you sandbox applications. The AI Gateway, by serving as the proxy for tool execution, allows exactly that:

- It provides a **curated set of tools** that an AI agent is allowed to use. The agent's capabilities are deliberately limited to the tools and operations that have been onboarded and vetted.
- It ensures every tool invocation goes through security checks. For example, an AI can't call the "SendPayment" API unless it has appropriate credentials and the content of the request is allowed.
- It logs all tool usage for audit. If an agent executed the "CreateUserAccount" action yesterday, we have a record of it – what was requested and what the result was.

In short, tool governance means *the AI can only "play in the yard" of approved tools, under watchful eyes*.

Model Context Protocol (MCP) Support

MCP is an open protocol designed to standardize how AI agents communicate with tools (also referred to as “plugins” or “servers” in this context). It defines a JSON-RPC interface over HTTP/SSE for tool usage, enabling a universal way for agents to call APIs. Microsoft and others are adopting MCP so that, for example, a Copilot agent can use a third-party plugin similarly across platforms.

Azure API Management has embraced MCP support, which significantly helps tool governance:

- **Exposing Existing APIs as MCP Tools:** APIM can take any REST API it manages and expose it as an **MCP server**. Practically, this means APIM auto-generates the JSON-RPC schema and endpoints so that an AI agent can call the API’s functions via a standard RPC call. For example, suppose you have an internal “HR System API” with operations like GetEmployeeInfo and CreateVacationRequest. APIM could present these as methods of an MCP server called “HRTools”. The agent doesn’t need to know the REST details; it just calls call("HRTools", "GetEmployeeInfo", {id:123}) via MCP, and APIM translates that to the actual REST request GET /employees/123 and returns the result. This can be done with **no custom coding** – APIM’s integration handles it.
- **Centralized Tool Security:** By fronting MCP tools with APIM, all calls to tools go through the API gateway. That means *central authN/Z still applies* (the agent likely uses a single identity to talk to APIM, and APIM then enforces which tools that token can call). It also means existing policies apply to tool calls just like normal API calls. If a certain tool should be rate-limited, you can apply APIM’s rate limit policy. If the input to a tool should be validated, you can write a policy for that. Essentially, **APIM treats tool invocations like any other API call**, folding tool governance into API governance.
- **Credential Injection and User Context:** Many tools require credentials. As mentioned, APIM’s credential manager can store those. In the MCP flow, APIM also supports the OAuth authorization mechanism defined in MCP – meaning if an agent needs to use a tool on behalf of a user, APIM can facilitate that user consent flow and store the resulting token. For example, a plugin that accesses a user's calendar might require the user to authorize it via OAuth. APIM can proxy that and then use the obtained access token when the agent calls the tool. The agent itself never handles the OAuth details; APIM does, ensuring **secure delegated access**.

- **Observability of Tool Use:** As with model calls, APIM logs every tool invocation and optionally its result. This gives a clear picture of **what the AI agents are doing**: which tools they are calling most, how often, with what inputs and outputs (if logging is enabled for payloads). From a governance perspective, this is gold. If an agent made changes in a system (like via an "UpdateRecord" tool), you have a log outside that system (APIM log) which records that the AI initiated it. This is important for audit – e.g., to answer, "who created this expense approval?" and find out it was the Finance Copilot agent on behalf of Alice, as logged at 3pm in APIM logs.

Governance Policies for Agent/Tool Use

We can set higher-level policies specifically around agent behaviors using tools:

- **Allowed Tool List:** The agent runtime (like Microsoft Foundry Agents) typically will only empower the agent with tools you specify. The gateway ensures it can't go beyond that, because any attempt to call a non-existent tool will fail.
- **Input Sanitization:** We might want to sanitize what an agent sends to a tool. For example, if an agent composes an SQL query to send to a database API, we could have APIM inspect that query string and ensure it doesn't contain a DROP TABLE or other destructive command (assuming the API unwisely allows raw queries). Ideally, the API itself controls that; but the gateway adds an extra layer of protection if needed.
- **Sequence Control:** One could ensure that certain sensitive tools can only be called in specific sequences or after certain other calls. For example, an agent must call a "Authorize" tool first to get a token, which APIM then requires to be present as a header for any subsequent data tool calls. APIM could enforce that pattern by rejecting any data tool call without a prior login call (tracked via a token or cookie it issues). This essentially forces the agent through a mini workflow that includes authentication steps.
- **Human Oversight for Sensitive Actions:** If an agent tries to perform a very sensitive action via a tool (say transferring a large amount of money via a finance API), the gateway could intercept and require a human confirmation. For instance, APIM could integrate with a logic that puts that request on hold and sends a notification to a human, expecting a special approval call or token in response. The agent would get a "pending" or error response in the meantime. This kind of human-in-the-loop gating can be implemented with custom policies or by orchestrating with other services (like an Azure Logic App that APIM calls to

initiate an approval workflow). While APIM doesn't have a built-in "pause workflow" feature, one could simulate this by having the tool itself designed to require an approval code (which the gateway obtains from a human).

All these are advanced scenarios, but the point is that with a gateway, *if you can think of a control, you can often implement it* with a combination of APIM policies, external integration, and careful design of how tools are exposed.

Universal Tooling Protocol – Future-proofing

By embracing MCP and centrally managing tools, the enterprise is future-proofing their AI integration in a few ways:

- **Compatibility:** If tomorrow a new AI agent platform emerges, that also speaks MCP, all your existing tools exposed via the gateway are immediately compatible with it. You won't need to rewrite your integration for that platform. MCP aims to be like *"the USB-C" of AI tools,* providing a standard connector. So, investing in exposing tools via MCP now (through APIM) means whatever agent frameworks come along (be it Microsoft, Google, AWS, etc.), you can plug them in with minimal effort.
- **Multi-Cloud and External Tools:** The gateway can manage not only internal tools but also interface with external ones. For example, if an external SaaS provides an MCP endpoint for its service (say a SalesForce CRM plugin), the enterprise might still choose to route through their gateway for monitoring. Or at least, if the agent is using it directly, that usage is registered in API Center for visibility. One could even create a pass-through in APIM if they wanted logs for those calls. So even in using third-party tools, the mindset remains: funnel as much as possible through governance points.
- **Extensibility:** As new types of tools arise (maybe async and real-time event-driven ones, or agent-to-agent collaboration endpoints), the gateway's job is to adapt and intercept where it can. When agents talk to each other via some protocol, the gateway pattern might extend to having a broker for those communications (maybe forcing agent messages through a central policy engine). We're not there yet, but having the gateway in the architecture means we already have a place to insert governance for new interaction types.

Future Trends: Agent-to-Agent Protocols

One current trend is agents that communicate with other agents (for delegation, specialization, etc.). For instance, one agent might ask another agent (possibly in a different system or organization) to perform a subtask. This introduces new governance questions: How do we ensure an internal agent doesn't inadvertently send sensitive data to an external agent? How do we audit what external info our agent consumed?

If these agent-to-agent interactions use open protocols (using MCP or something like Google's proposed A2A), then similarly to tools, the AI Gateway is in a good place intermediate them. For example, an enterprise might mandate that any communication between its agent and an outside agent goes through a gateway that strips or encrypts certain info and logs the interaction. Or they might set up gateway-to-gateway handshakes between organizations.

The gateway can add value in agent and tool discovery by integrating with registries and enabling access to those curated or authorized by enterprise security teams. It underscores that regardless of how AI systems evolve, having a **policy enforcement layer** (gateway) is valuable. It provides a chokepoint where you can implement controls or visibility as new patterns emerge.

Summary of Tool Governance

The AI Gateway, by using standards like MCP and centrally managing tool APIs, ensures that **AI agents remain under the enterprise's control even as they perform complex tasks**. It's akin to putting an AI agent in a secure sandbox: the agent can only do what it's allowed to do via the tools you've given it.

This way, organizations can confidently deploy AI agents that, say, automate business processes or interact with external systems, knowing that every external action the agent takes is *accounted for* and within policy. If an agent tries something outside its remit, the gateway will simply not allow it (or will flag it). Thus, the gateway mitigates the "automation risk" of agents by **governing their toolkit**.

> In practical terms, companies might maintain a **"tool registry"** with Azure API Center, where each entry has undergone security review, and the gateway is configured accordingly to let agents use that tool. The combination of careful curation and technical enforcement leads to trustworthy agent behavior.

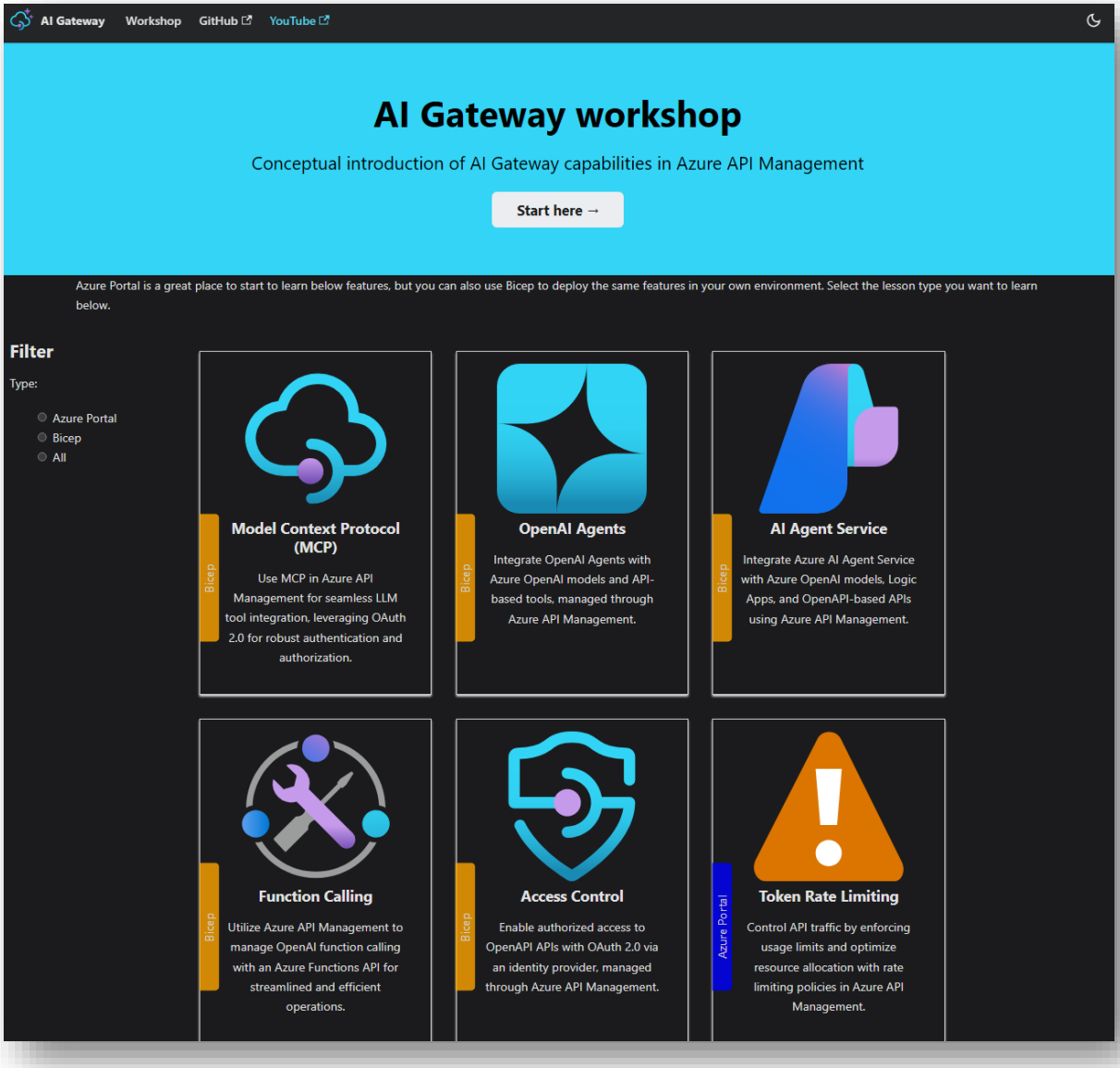
Chapter 6: Exploring the AI Gateway Capabilities

Thus far, we've discussed the AI Gateway conceptually and in terms of design. To make these ideas more concrete and to enable experimentation, Microsoft has provided an **open-source AI Gateway repository with samples and workshop content**. This resource allows practitioners to **explore common use cases and scenarios in sandbox environments**, with automation and one-click deployment of the necessary components.

The Open-Source AI Gateway Repo: Available on GitHub at aka.ms/ai-gateway, contains Infrastructure as Code, sample policies, test snippets, and configurations demonstrating Azure API Management's & Microsoft Foundry AI capabilities. It includes:

- **Labs with Scenario Examples:** For instance, you'll find examples for setting up **token rate limiting** , integrating **content safety** policy into a request pipeline, using **semantic caching** , and more. Each scenario has a guided tutorial trough a Jupyter Notebook. For example, the content safety notebook shows how to apply a content safety policy to an endpoint, send a test prompt that violates the policy, and observe AI Gateway blocking it. These walk you through enabling the feature and seeing the outcome, making the features tangible.
- **Automatic deployment:** Each lab provides Infrastructure as Code templates, primarily using Bicep, to fully automate deployment. With a single click (Run All option in the Jupyter Notebook), you get an APIM instance pre-loaded with the AI gateway sample configuration, plus Microsoft Foundry resources and project. The deployment also sets up any needed infrastructure like a cache for semantic caching and Azure Monitor for logging. The idea is to have a self-contained sandbox where you can immediately start calling the gateway and see policies in action, without manually provisioning everything. If you prefer Terraform over Bicep, you can use an existing Terraform lab or deploy the lab with Bicep and use the Export template feature to obtain the Terraform equivalent.

- **AI Gateway workshop:** The workshop enhances the labs by providing content that facilitates a hands-on learning experience directly through the Azure Portal.



Through these guided labs, you can see the gateway features working in a controlled environment. This is crucial because reading about token rate limiting conceptually is one thing but seeing an actual log entry of "promptTokens": 50 and "remainingTokens": 450 after a request really drives home how it works.

One-Click Sandbox Benefits: This sandbox environment allows teams to play with policies safely. They can test extreme cases (like sending huge prompts, or malicious

inputs) and see how the gateway handles them, without affecting any real production service. It's an environment to **experiment and learn**.

Additionally, since it's open source, users can contribute or adjust. If you want to try a custom policy not already in the samples (say, integrating with a different vector DB), you can modify the template or add a policy and share back your results.

Understanding Concepts in Practice: The labs likely cover scenarios drawn from real use cases – for example, implementing a gateway in front of Azure OpenAI to manage multi-application access (the scenario that was described as the reason these features were introduced). By doing it hands-on, architects and developers internalize *how* to configure and tune the gateway.

It also reveals any practical considerations: maybe how to monitor the gateway's performance, or how to hook it up with your CI/CD pipeline by reusing the Infrastructure as Code templates.

Adjusting Configuration and Policies: Once you have the sandbox and have tried the default settings, you can start **tweaking** them to simulate your own scenarios. For instance, you might change the token quota values and see how it behaves or incorporate an additional policy (like adding a custom header in responses to mark them as coming from the gateway, etc.). This is like a rehearsal stage for your eventual production rollout.

By experimenting in a low-risk environment, teams gain confidence. They can refine what combination of policies achieves their objectives (e.g., what's a reasonable token per minute limit that won't hamper legitimate use but will stop runaway loops? The lab data can help decide).

Preparing for Production Roll-out: After exploring and tinkering in the sandbox, the findings can feed into a production design. Perhaps you identify which metrics you want to monitor closely, or you decide to implement a particular fallback routing because you saw its effect. It's much better to discover and resolve issues in the lab than when the gateway is already in front of live services. The sandbox also can serve as a demo environment to get buy-in from stakeholders by showing the gateway's value.

> In summary, the open-source AI Gateway workshop provides a practical playground to solidify understanding, test scenarios, and adjust configurations – all of which helps ensure that when you deploy an AI Gateway in production, you do so with **confidence and clarity** about how it will behave.

Chapter 7: Implementing the Enterprise AI Gateway

After understanding the capabilities and best practices conceptually, the next step is **implementation** – setting up the Enterprise AI Gateway in a real-world environment. This chapter will outline how to weave Azure API Management, Microsoft Foundry, and related services into an enterprise architecture, using guidance from the Azure Well-Architected Framework and Azure Landing Zone blueprints .

Azure Well-Architected Framework Alignment: The Azure Well-Architected Framework provides pillars (Reliability, Security, Cost Optimization, Operational Excellence, Performance Efficiency) to consider applying for any solution. Implementing the AI Gateway should adhere to these principles:

- **Reliability:** Choose the appropriate APIM tier that supports high availability. For production, **Premium tier** is recommended because it supports multi-region deployment and zone redundancy. Deploy APIM across availability zones (or separate regions) so that even if one data center goes down, the gateway stays up. Use APIM's ability to scale units to handle peak load. Also, design backend model deployments for redundancy – for instance, have a primary and secondary model endpoint and configure APIM load balancing with priority routing (PTU first, then PAYG as fallback) as we discussed. Implement health monitoring – ensure APIM and model telemetry are fed to Azure Monitor and set up alerts for metrics like gateway availability, backend errors, or high latency.
- **Security:** Follow a *zero-trust* approach: place APIM inside a **VNET** (virtual network) if clients are internal, or behind an **Application Gateway (WAF)** if exposing to public clients. Many enterprise deployments put APIM in an internal VNET and then have an App Gateway or Front Door as a reverse proxy in the DMZ (with WAF rules) for any external traffic. Use **Microsoft Entra ID for client authentication** whenever possible (like using OAuth2 client credentials or JWT validation for internal apps). Leverage Managed Identities for APIM to call Microsoft Foundry so no keys are stored. Store any needed secrets (like third-party API keys) in **Azure Key Vault** and reference them via APIM named values (Key Vault integration) – that way, rotations happen centrally. Apply global policies for security headers and throttling to mitigate abuse. Also integrate APIM VNET with **Azure DDoS Protection** and use rate limiting to prevent misuse.
- **Cost Optimization:** Be mindful of APIM costs – Premium tier is pricier but needed for mission-critical. Use scaling judiciously; scale down in non-peak hours if possible. Monitor token usage metrics (via Azure Monitor) to identify costly patterns and optimize them – e.g., enable semantic caching to reduce token consumption, which directly lowers AI costs. Also, consider using **Reserved Capacity** for Azure OpenAI (if available) to get cost stability if forecasted usage is predictable. The gateway should enforce quotas to keep costs under control. Essentially, implement the FinOps measures: set budgets and alerts – these can tie into APIM's token metrics. If certain workloads can use smaller models, route them accordingly to save cost. Test throughput in a pre-prod environment to ensure APIM instance is not oversized (or undersized) – you want it "right-sized" for cost and performance.

- **Operational Excellence:** Treat the gateway configuration as code and automate as much as possible. Use the **Bicep templates** or Terraform equivalents to version-control API configuration and policies. This way changes go through pull requests, reviews, and can be deployed via CI/CD pipelines. Set up a development APIM instance where developers can safely test new policies with dummy backends (for example, test a new regex filter or new tool integration) before promoting to production. Use **APIM Workspaces** to allow teams to manage their changes in a collaborative way without blocking each other and improving ops. Also train the operations team on how to read the logs and metrics – e.g., know how to answer if a team asks: "why did our request get blocked?" or "are we nearing our quota limit?". Build **dashboards** (maybe using Azure Monitor workbooks or PowerBI) that the ops team and business owners can regularly check (like daily token usage by API, error rates, etc.). Plan backup/export of APIM config regularly (APIM has an API to get the configuration) so you have a snapshot, if something goes wrong or if you need to rebuild quickly (the IaC templates are your best friend here). Additionally, incorporate **automated tests** for the gateway policies – e.g., after deployment, run a set of test calls (maybe via Azure Load Testing) to verify critical policies are in effect (like ensure a blocked content prompt indeed returns the expected 403 or whatever).

Azure Landing Zone Blueprint: The Cloud Adoption Framework suggests using landing zones – a set of resources and configuration providing a secure, compliant baseline for workloads. For an **AI Gateway landing zone**, consider:

- Deploy APIM in its own resource group (or as part of an “Integration” resource group in a hub network). If using Microsoft Foundry, that service might be in a different resource group possibly under an “AI” category. Use **Resource Tags** to label these (env=prod, function=AI Gateway, etc.) for governance.
- Networking: in an enterprise setup most likely, you have a hub-and-spoke VNET topology. Place APIM (if internal) in the integration hub VNET. Connect Foundry and other services with private endpoints. Also, if APIM is internal but you have external clients (like a customer-facing app using the AI gateway), an Azure Application Gateway or Front Door in front will handle the external exposure. The blueprint should include that, if necessary, with WAF rules.
- Identity and Access: Register a **Microsoft Entra ID application** for the APIM Developer Portal (if using it) and for clients to request tokens. Set up an Azure AD group for internal developers allowed to publish APIs and give that group APIM admin roles (rather than managing individual accounts). A common best

practice is to use a service principal for the CI/CD workflows and keep the users with read-only access for the upper environments. Use **Managed Identity** features throughout (APIM's identity to backends, Foundry's identity for anything it calls, etc.). For on-premise or multi-cloud integration, if APIM needs to call something in AWS or on-prem, use **VNET integration or Azure VPN/ExpressRoute** accordingly.

- Monitoring and Logging: Enable **Azure Monitor Diagnostics** on APIM to capture Gateway Logs, LLM logs and Metrics. Ensure these logs flow into whatever central log system (maybe an Azure Monitor workspace integrated with your SIEM or to other systems leveraging the integration with Event Hub). The blueprint might specify creating an Azure Monitor Workbook for the AI gateway as a ready-made dashboard.
- Governance and Compliance: If needed, use **Azure Policy** to enforce certain rules on the APIM and related resources. For example, ensure the APIM is only accessible via HTTPS (which by default it is, but there is Azure Policy for it). Ensure resource locks or policies so that, for instance, no one can accidentally expose the APIM management endpoint publicly (if you use integrated auth). If required by compliance, turn on **Azure AD Privileged Identity Management** for APIM roles (so granting someone APIM admin is time-bound and approved).
- Pipeline: As part of a landing zone, define the **DevOps pipeline**: e.g., use GitHub Actions or Azure DevOps release pipeline to deploy APIM config (via Bicep/ARM/Terraform templates and/or automation scripts). Possibly incorporate integration tests in the pipeline (maybe calling a test endpoint on APIM post-deployment to verify it's working).

In sum, think of implementing the AI Gateway like implementing any enterprise platform: follow best practices for network isolation, identity management, monitoring, and Infrastructure-as-Code.

Many organizations also establish a **Center of Excellence (CoE)** when rolling out Enterprise AI initiatives, at scale. That CoE platform engineers might own the AI Gateway – meaning they set up the baseline config, define the onboarding process for new models/tools, monitor usage patterns, etc. The implementation should include establishing such operational processes. For example, decide how new APIs will be added: via pull request to the config repo, reviewed by a CoE member, tested in a staging environment, then deployed in production with confidence.

Best Practices Recap/Checklist:

- ❑ Use **APIM Premium** in production for scaling and redundancy.
- ❑ Place APIM in a secure network context, front with WAF for external access.
- ❑ Use **Managed Identities** for all backend access.
- ❑ Enforce **OAuth/JWT auth** for incoming calls (no anonymous access).
- ❑ Store secrets in **Key Vault**, not in APIM code.
- ❑ Set sensible **quotas/limits** from day one (even if high) to catch anomalies.
- ❑ Turn on **logging and metrics export** from day one and build workbooks and dashboards.
- ❑ Automate deployment and keep configs in **source control**.
- ❑ Implement at least one **non-production environment** (dev/test) that mirrors prod for safe testing.
- ❑ Continuously update as new APIM features or policies release (e.g., new LLM policies), evaluate and adopt them through the CoE.

Reuse the following **Landing Zones Accelerators** to jump start your AI Gateway implementation:

- Generative AI gateway scenario
- AI Hub Gateway Landing Zone

> Using the above approach and the landing zone methodology ensures the AI Gateway is not a one-off setup but an evolving, well-managed part of the enterprise architecture. It becomes a repeatable blueprint that can be rolled out across multiple environments (dev, test, prod) and even multiple business units if needed (with a central governance).

Chapter 8: Continuous Improvement

Building the AI Gateway is not a one-and-done task. Like any platform, it requires ongoing **continuous improvement** to adapt to new models, usage patterns, and business needs. This chapter covers how to use CI/CD, platform engineering, and automated evaluation to keep the AI Gateway and the AI services behind it at peak performance and value.

CI/CD for AI Gateway and Models

To rapidly iterate and safely roll out changes, use **Continuous Integration/Continuous Deployment (CI/CD)** pipelines:

- **Infrastructure as Code (IaC):** As mentioned, represent the gateway configuration (APIs, policies, named values, etc.) in code form (e.g., ARM templates, Bicep, Terraform). Author the IaC with AI assistant tools such as GitHub Copilot in combination with the template exporter from Azure Portal. Also treat Microsoft Foundry deployments and settings as code. Store the IaC in a git repository.
- **Automated Testing:** Whenever a change is made (like adding a new API or policy), run automated tests. For policies, you might use a script to deploy the change to a test APIM instance and run a battery of test requests with expected responses. For model changes, if it's just behind APIM, ensure APIM's routing still works, etc. If a new model is deployed (e.g., adding GPT-5 in addition to GPT-4.1), perhaps run a set of sample queries through both via APIM (in a test environment) to verify responses and performance.
- **Deployment Pipeline:** Use a pipeline (Azure DevOps or GitHub Actions for example) to deploy changes from dev → test → prod. For instance, if adding a new tool API: push changes to main branch triggers pipeline to deploy to dev APIM, run tests; if tests pass, a PR to merge to prod branch triggers deploy to prod APIM. Include approvals for prod deployment if required (especially in regulated industries).

- **Blue/Green or Canary Deployments:** For crucial changes, consider canary releases. APIM doesn't support canary in the same sense as code, but you can achieve similar by deploying a new revision (without breaking changes) or a new version (with breaking changes) of an API and only routing a small subset of traffic to it (via header or subscription). Or spinning up a parallel APIM instance for staging and doing test traffic. For model changes, perhaps initially route 10% of requests to the new model as trial (like we described in FinOps A/B tests). Manage this routing via APIM policies that use weight or conditional forwarding (maybe by a query param or a specific subscription for canary users).
- **Feedback Loop from Ops to Dev:** Use the logs from APIM to drive improvements. For example, if the ops team sees frequent policy violations or errors, feed that back so the dev team can adjust the policy or underlying service. Maybe refine a regex that's causing false positives or tighten a rule that isn't catching an issue.

By automating deployments, you reduce the risk of manual errors in complex APIM configs and ensure consistency. It also enables frequent updates; for example, if OpenAI releases a new model or APIM releases a new LLM policy, you can incorporate and deploy it swiftly through the pipeline.

Platform Engineering and Federated approach

The AI Gateway is essentially an internal platform offered to development teams (the ones building AI apps/agents). Platform engineering best practices are recommended:

- Provide **self-service** capabilities to teams. For example, if a team wants a new model onboarded, give them a way to request it (maybe via a pull request to the APIM config repo adding their API). Use APIM Workspaces so they can manage their changes (like their API or their product) without needing full admin. This decentralizes development but centralizes control.
- **Templatize common patterns:** The platform team can supply templates or examples for common policies (like "here's the standard content safety policy snippet we apply to all new APIs"). This consistency speeds up adding things and ensures governance is met automatically. Possibly use APIM policy fragments to ensure reusability and consistency or just copy from samples.
- **Documentation and Portal:** Keep internal docs for how teams can use the gateway (how to get access keys, how to interpret quota errors, etc.). Encourage the use of Azure API Center to discover what's already provided so they don't duplicate efforts.

Platform engineering also involves **observability for the platform itself**: monitoring deployment pipeline success, etc., and capacity planning (knowing when to scale up APIM units, or if you need to partition into multiple APIM instances for scaling vs. one global instance).

Automated Model Evaluation using Gateway Logs

One novel aspect of AI systems is evaluating model performance and choosing the best model for each task (often called "continuous model evaluation" or dynamic benchmarking). The AI Gateway can facilitate this:

- **Logging for Benchmarking:** Log **which model produced which output** along with any available quality signals (user feedback, or a downstream metric). Over time, by accumulating this data, you might find, for example, that Model A's outputs had to be edited by users 20% of the time, whereas Model B's only 10% – indicating Model B is better for that use case despite cost. The gateway logs make it easier to do this comparison.
- **Periodic Reruns:** You can use the logged prompts to periodically re-run on different models. For instance, weekly, take a sample of prompts from production (maybe last week's 1000 queries) and run them through a candidate new model (in a non-prod setting). Next, compare the outputs with those from the production model to determine whether the new model performs as well or better. A platform team could automate this by exporting prompt logs and feeding them to a model evaluation process (using frameworks such as **Azure AI Evaluation SDK**).
- **Adaptive Routing:** In conjunction with A/B testing, one could use performance data to adjust APIM's routing weights automatically. For example, if after a trial period Model B consistently performs better (e.g., fewer errors or higher user ratings captured externally), you might update APIM to make Model B primary. This might still be a manual decision assisted by data.
- **Ensure Best Model per App:** Perhaps some apps benefit from a smaller model (because their queries are simple) and others truly need the largest reasoning model. Use logs and metrics to cluster usage by application and outcome. You might find "Team X's questions to the bot are always short, they could use gpt-5-nano model instead of gpt-5 and still be happy". Then you can create a new API endpoint for that team using a cheaper model, saving cost. Consider using the Foundry model router 🔗 for automatic model selection based on the prompt.

In essence, treat the model behind the gateway as something that can be **optimized continuously**. The gateway gives the abstraction layer to swap things under the hood and the data to know whether to do so. This is analogous to how a database team might tune indexes or how a cloud ops team might move workloads to cheaper instances – here we tune AI models for cost/performance.

Continuous Learning and Tuning

Beyond technical measures, continuous improvement means regularly analyzing how the gateway is used:

- **Review Logs and Analytics Regularly:** Have quarterly (or monthly) reviews where the CoE looks at the token usage trends, most used tools, any anomalies or errors. Use that to adjust quotas, capacity, or maybe provide additional training to teams (e.g., if many unsafe prompts are attempted, maybe users need education).
- **Stay Updated with Azure Improvements:** Azure will update APIM and Foundry with new features and better synergies. E.g., new policies, new monitoring capabilities, etc. Incorporate those by monitor the API Management release notes and announcements .
- **Feedback from Users:** Seek feedback from the app developers using the gateway: Is the latency acceptable? Are the errors understandable? If the gateway blocks something, does the error message tell them why (you can customize error messages via APIM policies)? Improve those things so that the platform is developer-friendly and meet business needs.
- **Model Lifecycle:** Manage the lifecycle of models: retire those that are obsolete (and use APIM's API version deprecation features to communicate to app dev's, perhaps by adding a warning header or email). Onboard new models after evaluation. This is akin to a DevOps process but for models: monitor if a model's performance drifts (maybe OpenAI models improve or change behavior with time; logs might capture if outputs are getting longer or different, affecting cost or quality).
- **Continuous Security Testing:** Since new threats may emerge (jailbreaks, prompt injection attacks, etc.), regularly test the gateway's defenses. Simulate attacks in a safe environment to see if the content filters and policies hold up. If not, update them. Consider Microsoft Defender for APIs for full lifecycle protection, detection, and response coverage for APIs.

By embracing continuous improvement, the AI Gateway will not stagnate. It will adapt to handle more load as adoption increases, incorporate better models as they launch, tighten or relax policies based on real usage evidence, and overall remain an enabler rather than a roadblock for AI innovation.

SRE Agent for the AI Gateway

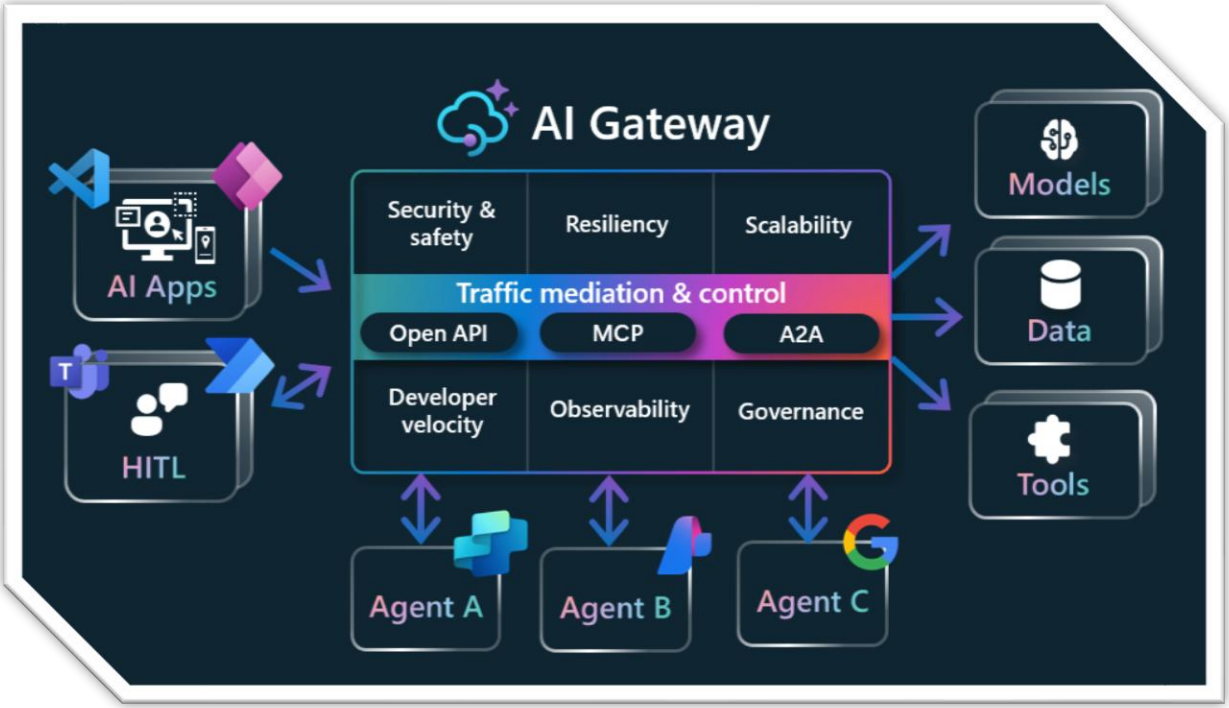
Microsoft has expanded the capabilities of the Azure SRE Agent, an AI-powered Site Reliability Engineering assistant, to now support Azure API Management (APIM). This enhancement provides:

- Proactive Monitoring: Tracks metrics like CPU usage, latency, error rates, and availability to detect anomalies early.
- Backend Visualization: Maps API operations to backend services such as AI Models, or tools, and visualizes their health.
- Root Cause Analysis: Correlates backend issues (e.g., memory errors, timeouts) with API failures.
- Configuration Diagnostics: Identifies misconfigurations such as malformed policies or incorrect scaling settings.
- Intelligent Remediation: Can suggest or apply fixes automatically with user approval.

> To summarize Chapter 8: use modern DevOps practices to deploy and update the gateway and underlying models frequently and reliably; use platform engineering to balance central governance with team-level agility; and leverage the rich data the gateway provides to continuously tune both the cost and performance of your AI systems. This creates a virtuous cycle: data from the gateway informs improvements, improvements are rolled out via DevOps, which then yield new data, and so on.

Chapter 9: Governing AI Agents

So far, we've addressed governing the components that AI agents use – models, data, tools. Now, we turn to **governing the AI agents themselves**. An AI agent (like a multi-step reasoning entity, for example a Copilot that plans tasks or an autonomous research agent) introduces additional considerations because it has more complex behavior than a single prompt-response. Here's how the AI Gateway concept plays a key role in securing, observing, and controlling agents, and what future developments might look like.



The Role of the AI Gateway for Agents: Think of an AI agent as an orchestrator that might chain multiple model calls and tool calls to accomplish a task. The gateway can be seen as the *policy enforcement point for every action the agent takes*. It **secures, observes, and controls** those actions:

- **Guardrails:** By using the gateway for its model and tool calls, we ensure the agent cannot step outside approved bounds. E.g., if an agent's prompt tries to get it to call an unapproved tool, the gateway simply doesn't have that tool endpoint – the agent will get an error or no action, effectively containing it.

Similarly, if an agent tries to generate disallowed content, the content safety policies at the gateway will intercept it.

- **Visibility:** The gateway provides full visibility into agent activities and outcomes. Without a gateway, agents would be a black box – you ask them to do something, and they might call various internal MCP tools or external APIs without any trace. With the gateway, every external interaction is logged. This addresses a top concern in AI agent adoption: *lack of visibility*. Many leaders worry "what if the agent does something and we have no idea?" The gateway mitigates that by capturing a lot of it.
- **Unified Control:** Enterprises likely will have multiple agents (marketing assistant, IT automation agent, etc.). The gateway allows a **central governance layer across all these agents**. The alternative would be each agent platform implementing its own controls – which may be inconsistent. Having a gateway means even if one agent is built with Microsoft Foundry, another with Logic Apps Agent Loop and another with CrewAI, they all go through the same choke points. This provides a consistent enforcement of rules.

Key governance needs for AI Agents (as identified by enterprise customers), include **Control**, **Compliance**, **Security**, and **Monitoring** of agents. Let's see how those are addressed:

- **Control (Policies & Boundaries):** Ensure agents operate within clearly defined boundaries. For example, define which data sources they can access, which actions they can take autonomously vs. require approval. The gateway is where these boundaries are technically enforced (tools available, content filtering, etc.), but also one should configure the agent itself with constraints (like prompt instructions that it should not do X). Governance means both configuring the agent's prompt (soft guardrail) *and* having the gateway (hard guardrail) to cover if the agent deviates or the prompt is bypassed.
- **Compliance & Oversight:** Maintain visibility into agent activities and have auditing. The gateway provides logs that compliance officers can review. If every action is logged, you can audit an agent's work similar to a human's – e.g., an agent closed a customer ticket at 2pm, here's the transcript and tools used. If an agent misbehaved, you have forensic data to analyze how and why. Some organizations are starting to mandate that AI activities be auditable; a gateway makes that feasible.
- **Security & Access Control:** Implement strong permission management for agents. This means making sure an agent only accesses what it should. You don't give an agent a god-mode API key to everything. Instead, you might give it

an identity with specific roles, and the gateway enforces those roles on each attempted action. For example, if an agent acting for an HR user tries to access a finance database tool, the gateway can say “no, your token doesn’t have access to FinanceAPI”. Also, ensure agents are not using tools in unintended ways (like feeding sensitive data into a third-party search). We can also incorporate **Threat Protection**: using Azure Defender for APIs or other anomaly detection on the agent’s traffic, perhaps to catch if an agent is compromised or manipulated (this is emerging tech, but conceptually one could detect if an agent suddenly starts calling an API excessively, etc.).

- **Agent-to-Agent Interactions:** MCP allows agents to talk to other agents or systems directly (without going through the central orchestrator). The gateway could be extended with specialized “Agent Policies” that handles inter-agent communication, applying trust rules (like Enterprise agent will only talk to agents from orgs that are in a trusted list, and remove sensitive info from its messages). This is analogous to email filtering for human communications. We mention this to highlight that governance is an evolving target – but having a mindset and platform ready (the current gateway) means you’d likely extend it to those scenarios too.
- **Promoting Trusted Agents:** In governance terms, enterprises will likely differentiate between “blessed” agents (ones built and vetted by the CoE) and ad-hoc or external ones. The *policy governance* can enforce that only “trusted agents” get certain privileges. For example, a personal agent someone created might be prevented (by lack of credentials) from accessing sensitive internal APIs, whereas an officially sanctioned agent has a client ID recognized by the gateway as allowed more scope.
- **Responsible AI Toolkits:** Tools like the OpenAI Function calling, MCP, etc., are making it easier to structure agent actions, which ironically also makes it easier to intercept and govern them (because they become API calls). We might see more fine-grained policy – e.g., “disallow agent from using Tool X with Parameter Y > 1000 because that could be dangerous.” As descriptions of tools improve (with metadata about what they do), automated governance might become possible (like a rule engine reasoning "this tool writes to production DB, only production-approved agent can call it").

In all this, the **AI Gateway remains fundamental**: it's the "choke point" where you can intercept and enforce **governance frameworks**.

So, governing AI Agents = ensuring **guardrails, observability, and control** for the overall agent behavior, not just individual calls. The combination of careful agent design (prompt constraints, user permission checks in agent code) and the external enforcement (gateway) is key.

Summary: The AI Gateway secures AI agents by controlling their inputs and outputs (models, data, tools), providing full visibility into their "thoughts" (at least the externalized ones), and applying organizational policy to every action. Advancements such as agent-to-agent communication will introduce new challenges, but the pattern of using gateways or brokers to manage those communications will likely follow.

By doing this, enterprises can allow AI agents more autonomy *safely*. They can let an agent execute multi-step workflows that save humans time, while still having the confidence that there are brakes and logs – essentially, an **AI governance safety net** – in place. This ensures AI agents are powerful helpers, not loose cannons, aligning with the company's risk tolerance and ethical standards.

Chapter 10: Conclusion

We have journeyed through the motivation for an Enterprise AI Gateway and its many facets – from **managing model usage** with fairness and cost-control, to **governing data access** and ensuring compliance, to **sandboxing tools** for safe AI augmentation, and finally to overseeing holistic **agent behavior**. Throughout, a consistent message emerged: *AI's transformative power must be balanced with robust governance*.

The **AI Gateway** – embodied by Azure API Management plus Microsoft Foundry and related services – is the centerpiece of that balance. It allows enterprises to **innovate with AI confidently**. Rather than being a roadblock, it's an enabler with guardrails: teams can rapidly onboard models and tools because the gateway handles security, scalability, and monitoring automatically. Business units can experiment with AI knowing that costs won't spiral out of control and any issues can be traced and addressed.

Key takeaways and inspiring closing thoughts:

- **AI is incredibly powerful**, but with great power comes great responsibility. Unchecked AI usage can lead to data leaks, customer harm, or runaway costs. The AI Gateway is the mechanism by which we **take responsibility** – it ensures AI is used **for good, within set boundaries**.
- **Every AI enterprise architecture will need an AI Gateway**, just as today every enterprise has network firewalls and API gateways. It becomes a fundamental layer: as AI becomes ubiquitous (in apps, in employee-facing copilots, in autonomous workflows), a centralized governance layer is the only practical way to enforce consistent policies and learn from enterprise-wide AI data.
- **Continuous evolution:** The field of AI is moving fast. New models, new threats, new regulations will appear. The AI Gateway approach is forward-looking – it gives a structure to incorporate future advancements. For instance, if regulators tomorrow require logging of all AI decisions, those using an AI Gateway are already a step ahead (logging is built-in). If a revolutionary model arrives, it can be plugged in behind the gateway rather than chaos for app devs. In that sense, the AI Gateway future-proofs an organization's AI strategy.
- **Ethical AI and Gateway:** There's a human and ethical dimension as well. We want AI systems to behave **responsibly and reliably**. The gateway helps enforce

responsible AI practices (like content moderation, privacy protection) uniformly. It is a tangible implementation of principles (such as Microsoft's Responsible AI principles) into running systems. This means as AI continues its breakneck advancement, the enterprise can harness it while **upholding trust and compliance**.

To conclude, we return to the core vision: *AI has the potential to augment every aspect of business, from customer service to decision making*. By implementing an AI Gateway, enterprises ensure that this augmentation happens under **control, with clear visibility and aligned with organizational values**. It transforms AI adoption from a wild west into a well-governed ecosystem.

Just as APIs enabled digital transformation (and API Management became essential), AI will enable a new transformation – and the **Enterprise AI Gateway will be its essential guardian and enabler**.

Inspire & Empower: With the AI Gateway in place, organizations can embrace AI's possibilities fully – allowing more projects to incorporate LLMs and agents, because there is confidence in oversight. This democratizes AI within the company: teams that were hesitant due to risk can proceed, knowing the gateway has their back on safety and compliance. It creates an environment where **innovation and governance go hand-in-hand**.

Ultimately, the hope is that AI, governed well, will amplify human potential, drive efficiency, and unlock insights in a way that is safe, secure, and equitable. The AI Gateway is a fundamental component to achieving that hope in reality. It lets us say "yes" to AI – not with fear of the unknown, but with **excitement for the future** and the safeguards to ensure it benefits everyone.