BUILDING ENTERPRISE AI AGENTS ON MICROSOFT · PLAYBOOK

An agent perceives, decides, and acts, bounded by policy.

Enterprise AI agents are not chatbots. They perceive, decide, and act, with autonomy bounded by policy. This playbook maps the three Microsoft platforms, Copilot Studio, GitHub Copilot, and Azure AI Foundry, the four agent types and the six-layer architecture, the Human-in-the-Loop guardrails that scale, two production case studies, and a twelve-month adoption roadmap, all seen through GitHub Copilot.

AUTHOR

Paula Silva

ROLE

AI-Native Software Engineer

CONTACT

in/paulanunes

READING TIME

~ 45 minutes, end to end

3
MICROSOFT PLATFORMS

6
CHAPTERS

4
AGENT TYPES

5
ADOPTION PHASES

CHAPTERS

Chapter 00

What an agent is, and is not

The working definition, perceive, decide, act, bounded by policy, chatbot versus agent, the five components every agent shares, and the four-layer decision stack.

Foundations →

Chapter 01

Three platforms, one framework

How to choose between Copilot Studio, GitHub Copilot, and Azure AI Foundry by audience, skill level, time-to-pilot, and governance model.

Platforms →

Chapter 02

Types and the six-layer architecture

The four enterprise agent types matched to the work, the six layers from governance up to the surface, MCP tools, model choice, and RAG done right.

Architecture →

Chapter 03

Human-in-the-Loop governance

The four HITL quadrants by impact and reversibility, three implementation patterns, and how to reuse Azure governance instead of parallel controls.

Governance →

Chapter 04

Production case studies

Vehicle leasing in production: four agents against a 15-day fine deadline, three agents for NF-e tax compliance, and the ingest, classify, act, escalate pattern across four verticals.

Case study →

Chapter 05

Adoption roadmap

The five-phase roadmap over twelve months, Discover to Operate, explicit exit criteria per phase, and cost tracked per loop in GitHub AI Credits.

Roadmap →

READING TIPS

KEYBOARD SHORTCUTS

Press `/` to search. Use `j` and `k` to move between chapters. Press `t` to toggle theme. Press `g` to go to top.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

00 AGENT ANATOMY

What an AI agent is, and is not: perceive, decide, act, bounded by policy.

An AI agent perceives, decides, and acts, and its autonomy is bounded by policy. That one sentence separates today's agents from yesterday's chatbots, and it sets up everything that follows. This chapter gives the working definition, contrasts a stateless chatbot with a goal-driven agent, names the five components every agent shares, and places them in a four-layer decision stack, all seen through GitHub Copilot.

Perceive, decide, act: the working definition, bounded by policy

An AI agent is software that perceives, decides, and acts. Its autonomy is bounded by policy. That definition is short on purpose, and every word carries weight. Perceive means the agent reads its input: a prompt, a file, a repository, an event, the result of its last action. Decide means it plans the next step toward a goal, rather than returning a single canned reply. Act means it changes something in the world: it edits a file, calls an API, runs a command, opens a pull request. The line between what came before and what enterprises deploy today runs exactly through those three verbs.

The fourth clause is the one teams forget. Autonomy is bounded by policy, and the boundary is part of the definition, not a control you bolt on afterward. GitHub Copilot in agent mode makes this concrete. It reads your repository, plans a change, edits files, runs the build, and opens a pull request, and it does all of that inside the limits you write in .github/copilot-instructions.md. That instruction file is where the policy lives: what the agent may touch, which conventions it must follow, when it must stop and ask. An agent with no policy is not more capable. It is unsupervised.

> DEFINITION
>
> An AI agent perceives, decides, and acts, with autonomy bounded by policy. The platform you pick sets the boundary. The architecture you ship earns the trust.

Chatbot versus agent: stateless and reactive versus goal-driven, stateful, and tool-using

Yesterday's chatbot was stateless and reactive. It received a question and returned a single answer. It held no memory across turns, it could not call tools, and it had no notion of pursuing a goal. Ask it twice and it starts fresh each time. That design was fine for frequently asked questions and scripted support flows, and it is still fine for those. It is not an agent, and calling it one sets the wrong expectation.

Today's agent is goal-driven, stateful, and tool-using. It receives a goal, plans a path to reach it, calls external tools, observes the results, and iterates until the goal is met, or escalates to a human when it cannot proceed safely. The difference is not the model. The same model in the Copilot picker can power a one-shot answer or a multi-step agent. The difference is the loop around it and the state it keeps between steps. When GitHub Copilot moves from a chat reply to agent mode, the model does not change; the loop, the memory, and the tool access do.

The five components every agent shares: Brain, Memory, Tools, Policy, Orchestration

Every agent, regardless of platform, is built from five components. The Brain is the model that reasons over the input and plans the next step, picked per task. In GitHub Copilot the Brain is whichever model you select in the Copilot picker: Claude Opus 4.8 for deep reasoning, Claude Fable 5 or Sonnet 5 for balanced work, Haiku 4.5 for fast and cheap turns, Gemini 3.1 Pro, or a GPT-5.x model. You choose per task, not once for the whole program. The Memory is the conversation history, a working scratchpad, a vector store for retrieval, and episodic memory across runs. Memory is bounded by policy: an agent remembers only what it is allowed to remember.

Tools, Policy, and the loop that ties them together

The Tools are the functions, APIs, database queries, code execution, and workflow triggers the agent can call. They are catalogued and scoped through the Model Context Protocol: each MCP server is a versioned, named set of tools with a defined boundary, so the agent never talks to a random endpoint. The Policy is RBAC, content filters, rate limits, Human-in-the-Loop checkpoints, and audit trails, and it is non-negotiable; in GitHub Copilot the .github/copilot-instructions.md file and your organization rules carry it, and Azure AI Content Safety enforces the content filters. The Orchestration is the plan-act-observe loop, multi-agent coordination, retries, and timeouts, and it is the platform's responsibility. Brain reasons. Memory persists. Tools act on the world. Policy bounds. Orchestration drives the loop.

The decision stack as mental model: anatomy, then platform, then type, then Human-in-the-Loop

Hold those parts in a four-layer decision stack, because each layer constrains the one below it. Layer I is Anatomy: perceive, decide, act, and the five components. Layer II is Platform: Copilot Studio for citizen developers, GitHub Copilot Agents for engineering teams, Azure AI Foundry for regulated production. Layer III is Type: development, productivity, business process, or data and analytics, matched to the task family and the persona that owns it. Layer IV is Human-in-the-Loop: four quadrants set by impact and reversibility, from full autonomy to two-person sign-off.

Read the stack from the bottom up, and build in that order. Get the anatomy right first, because a platform choice cannot fix a broken definition. Then pick the platform, then match the agent type to the work, then set the Human-in-the-Loop boundary that the action's risk demands. Skip a layer and the agent reaches production unsupervised. The pressure to skip is real: Gartner projects that 40% of enterprise applications will feature task-specific AI agents by 2026, up from less than 5% in 2025. Anatomy is the layer you cannot skip. Everything later in this playbook rests on it.

References

Primary sources for the working definition, the five components, and the platform and protocol references used throughout this chapter.

- GitHub Copilot Documentation, the reference platform for agent mode, custom agents, and the copilot-instructions file used throughout this playbook.
- Model Context Protocol Specification, the open standard that catalogues and scopes the tools an agent can call.
- Microsoft Copilot Studio Documentation, the low-code platform for business users and citizen developers.
- Azure AI Foundry Documentation, the platform for multi-agent orchestration and production-grade agents.
- What is Azure AI Content Safety?, the content filters that enforce part of an agent's policy layer.
- Gartner, task-specific AI agents in 40% of enterprise apps by 2026, the adoption projection: 40% of enterprise applications will feature task-specific AI agents by 2026, up from less than 5% in 2025.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

01 PLATFORM CHOICE

Three platforms, one decision framework: Copilot Studio, GitHub Copilot, and Azure AI Foundry.

Microsoft ships three platforms for building AI agents: Copilot Studio, GitHub Copilot, and Azure AI Foundry. They are complementary, not competitive, and most enterprises run all three. The hard question is never which platform wins. It is which use case lands where. This chapter gives you a decision framework built on four axes: audience, skill level, time-to-pilot, and governance model.

Three platforms, three audiences

Copilot Studio targets business users and citizen developers. It offers drag-and-drop topic authoring, native connectors to Microsoft 365, Dataverse, and Power Platform, and a deployment path measured in hours to days. Someone who has never opened an IDE can ship a working agent, and that is the point. Copilot Studio fits when the builder sits closer to the business process than to the codebase, and when the agent lives inside Microsoft 365 rather than inside a repository. HR self-service, IT helpdesk automation, and customer-service bots are its natural home.

GitHub Copilot targets software engineers working inside the IDE and the repository. Agent mode reads the code, plans a change, edits files, runs commands, and opens a pull request, all inside the developer workflow. Azure AI Foundry targets ML engineers and platform teams. It provides the full SDK, a model catalog, multi-agent orchestration, fine-tuning pipelines, and production observability. Time-to-pilot on Foundry runs in weeks, but the ceiling is far higher. The three platforms are complementary. Copilot Studio owns the conversational surface, GitHub Copilot owns the engineering loop, and Azure AI Foundry owns the vertical agents that carry strict SLAs.

Choosing by audience, skill, time-to-pilot, and governance

Four axes decide where a use case lands. Audience comes first: business and citizen developers point to Copilot Studio, software engineers point to GitHub Copilot, and ML or platform teams point to Azure AI Foundry. Skill level is second: Copilot Studio is low-code and no-code, GitHub Copilot is code-first and IDE-native, and Azure AI Foundry is pro-code with a full SDK. Read those two axes together and most decisions make themselves. The builder and the build surface already tell you which platform to open.

Time-to-pilot is the third axis. Copilot Studio ships in hours to days, GitHub Copilot in days, and Azure AI Foundry in weeks. Governance is the fourth, and teams underrate it. Copilot Studio inherits Power Platform administration, GitHub Copilot inherits GitHub policies, and Azure AI Foundry inherits Azure RBAC and Content Safety. Pick the platform whose native governance already matches the risk of the use case. Retrofitting controls after the pilot costs more than choosing correctly on day one. Gartner predicts 40% of enterprise apps will feature task-specific AI agents by 2026, up from less than 5% in 2025, so the volume of these decisions is about to rise.

GitHub Copilot agent mode in the repository

Agent mode is where GitHub Copilot stops suggesting lines and starts doing work. It reads the repository, plans a change, edits files, runs commands, and opens a pull request under review. The behavior is shaped by one file: .github/copilot-instructions.md. That file carries the conventions the codebase already encodes, exception handling, logging format, test organization, and naming, so every agent mode session inherits them instead of guessing. Write it once and every task starts grounded. MCP tools extend the agent beyond the repository. Each Model Context Protocol server is catalogued, versioned, and scoped, giving the agent access to build systems, ticket trackers, and internal services without ever talking to a random API.

The Copilot model picker turns model choice into a per-task decision. Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro, and GPT-5.x all sit in the same picker, and you route the task to the model that fits its cost and difficulty. Since June

1, 2026, premium model usage bills as GitHub AI Credits, where 1 credit equals $0.01. Every agent mode session and every background Coding Agent task is a metered line item. Treat those credits as infrastructure spend, budget them per team, and the FinOps story stays honest.

> DEFINITION
>
> Agent mode grounds itself on .github/copilot-instructions.md and reaches the outside world through scoped MCP servers. Model choice is a per-task decision in the Copilot picker, and every premium call is metered as GitHub AI Credits at 1 credit = $0.01.

Where each use case lands

Map the four axes onto real work and the boundaries get sharp. HR self-service and IT helpdesk automation land on Copilot Studio: the builder is a business owner, the connectors to Microsoft 365 are native, and the pilot ships in hours to days. Code generation, refactoring, PR automation, and test generation land on GitHub Copilot agent mode: the work lives in the repository, the conventions live in .github/copilot-instructions.md, and the pull request is the checkpoint. Neither of these use cases needs a full SDK, and forcing them onto one wastes weeks.

Regulated vertical agents land on Azure AI Foundry. A finance KYC and AML flow, a healthcare prior-authorization flow, or a Brazilian NF-e tax flow with a 15-day legal deadline all demand multi-agent orchestration, Azure RBAC, Content Safety, and dual-control Human-in-the-Loop on the irreversible actions. That is the ceiling only Foundry provides, and the weeks of time-to-pilot buy the SLAs the vertical requires. The rule that holds across all three platforms is the same: start narrow. One use case, one agent, one team, and generalize only after the value is proven.

One estate, one framework

Most enterprises will not pick one platform. They will run Copilot Studio, GitHub Copilot, and Azure AI Foundry side by side, each carrying the use cases it fits. The framework is what keeps that estate coherent: the same four axes, applied per use

case, so every agent lands on the platform whose audience, skill level, time-to-pilot, and governance already match it.

References

Primary sources for the three platforms, their governance model, and the agent integration patterns in this chapter.

- Microsoft Copilot Studio documentation, the low-code platform for business users and citizen developers.
- GitHub Copilot documentation, the reference for agent mode, .github/copilot-instructions.md, MCP tools, and the model picker.
- Azure AI Foundry documentation, the pro-code platform for multi-agent orchestration and regulated vertical agents.
- What is Azure AI Content Safety?, the governance control that Azure AI Foundry inherits for regulated workloads.
- Model Context Protocol, the open standard that scopes and catalogues the tools an agent can call.
- Gartner, task-specific AI agents in 40% of enterprise apps by 2026, the market forecast behind the volume of platform decisions ahead.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

02 METHOD ARCHITECTURE

Agent types and the six-layer architecture: match the work, then stack the platform.

An enterprise agent program has two design decisions to get right before any code ships. The first is the agent type, which follows from the work and the persona that owns it. The second is the architecture, six layers stacked from governance up to the surface where a user meets the agent. This chapter defines the four types, walks the six layers from the ground up, and shows where tools, models, and context each belong.

Four agent types, matched to the work

Four agent types cover most enterprise demand, and each maps cleanly onto a platform. Development agents live in the repository and the IDE. GitHub Copilot in agent mode is the natural home, with repository instructions in .github/copilot-instructions.md and MCP tools wired to build systems and issue trackers. Typical tasks are bug fixes, refactors, and test generation. Productivity agents operate inside Microsoft 365. Copilot Studio handles custom topic authoring, and Microsoft 365 Copilot Agents deliver embedded experiences in Teams, Outlook, and SharePoint. Typical tasks are meeting notes, email drafts, and document search.

The other two types carry more risk. Business process agents orchestrate across systems. Copilot Studio owns the conversational layer, and Azure AI Foundry takes over when the workflow needs strict SLAs, multi-agent coordination, or fine-tuning. Typical tasks are employee onboarding, approval workflows, and claims processing. Data and analytics agents ground their answers on enterprise data held in Fabric, Synapse, or a custom warehouse. They demand strict prompt engineering, a rigorous evaluation harness, and human sign-off before any output is published. Typical tasks are natural language to SQL and insight summaries. The type is not a label. It sets the governance you owe the agent.

The six layers, from the ground up

Production agent programs are built in six layers, and each layer constrains the one above it. The base is Platform and Governance: identity, RBAC, Content Safety, audit, observability, and FinOps. These are the same controls that already run the rest of Azure, not a parallel stack. Above it sit Models: a catalog of models picked per task, fine-tuned only where the payoff justifies the cost. Above Models is Context and RAG: a vector store, semantic retrieval, and identity-scoped access to enterprise data. These three layers decide what the agent knows before it acts.

Tools, orchestration, and the surface

The top three layers decide what the agent does. Tools and MCP is the registry of catalogued, versioned, and scoped MCP servers and plugins that let the agent act on the world. Orchestration is the runtime: the plan, act, observe loop, multi-agent coordination, retries, HITL checkpoints, and evaluation. Surface is where users meet the agent, in Teams, Outlook, the IDE, or a custom app. Skip a layer and the gap does not disappear. It moves to production, unsupervised. Start narrow, prove one use case on all six layers, then generalize.

> DEFINITION
>
> Six layers, from the ground up: Platform and Governance, Models, Context and RAG, Tools and MCP, Orchestration, and Surface. Each layer constrains the one above it. Skip one and the agent reaches production without the control that layer was supposed to enforce.

Cataloguing tools through MCP

Tools are where an agent stops talking and starts acting, so this is the layer that needs the tightest discipline. Every tool an agent can call is a function, an API, a database query, or a workflow trigger, and each one is exposed through the Model Context Protocol. MCP is the open standard for how an agent discovers and calls tools, and it turns an ad hoc set of integrations into a governed catalog. The rule is

simple. No agent talks to a random API. Every tool is registered in an MCP server registry, versioned, and scoped to the smallest permission that does the job.

The catalog carries three obligations. Register every server so a platform team can see what exists and retire what drifts. Scope every token so an agent holds only the access its task requires, never the caller's full rights. Audit every call so a reviewer can reconstruct what the agent did and why. In GitHub Copilot, the same MCP servers back agent mode in the IDE, which means a development agent and a business process agent draw tools from one governed source rather than two divergent ones. A catalogued tool is reviewable. An uncatalogued one is a liability waiting for an incident.

Picking the model per task from the Copilot picker

The model is a choice, not a default. In GitHub Copilot the picker exposes several models, and the right one depends on the task in front of you. Claude Fable 5, Opus 4.8, Sonnet 5, and Haiku 4.5 sit alongside Gemini 3.1 Pro and the GPT-5.x family. A short refactor with a tight latency budget does not need the largest reasoning model. A multi-file migration with deep dependencies does. Match the model to the difficulty of the task, and re-check the match when either the task or the model changes.

Cost is part of the choice. GitHub Copilot bills model usage in GitHub AI Credits, where one credit equals one cent, in effect since June 1 2026. A larger model spends more credits per request, so the picker is also a budget decision, not only a capability one. Treat every model swap as a change that needs proof. Keep a golden test set per agent, run regression on every swap, and hold a latency budget per loop. A model that passes the eval earns the traffic. A model that does not stays out, whatever its reputation.

Context and RAG done right

The Context and RAG layer decides what the agent knows before it reasons. Done well, it grounds answers on enterprise data and cuts the hallucinations that come

from a model guessing at facts it was never given. Three pieces make it work. A vector store holds embeddings of the enterprise corpus, from documents to code to tickets. Semantic retrieval pulls the passages that actually bear on the request, not the ones that merely share keywords. And identity-scoped access enforces that the agent retrieves only what the calling user is allowed to see.

That third piece is the one teams skip, and it is the one that matters most in a regulated estate. Identity-scoped retrieval means the vector store honors the same RBAC that governs the source systems. An agent acting for a support engineer must not surface a document that the engineer could never open directly. Wire retrieval to the identity layer from the Platform and Governance base, not around it. A data and analytics agent that grounds on Fabric or Synapse still owes a human sign-off before it publishes, because grounding lowers the hallucination rate but does not drive it to zero.

References

Primary sources for the platforms, the tool protocol, and the governance controls this chapter uses.

- Microsoft Copilot Studio documentation, the platform for productivity and business process agents built by citizen developers.
- GitHub Copilot documentation, agent mode, repository instructions, and the model picker for development agents.
- Azure AI Foundry documentation, the model catalog and multi-agent runtime for vertical agents with strict SLAs.
- What is Azure AI Content Safety?, a control in the Platform and Governance layer at the base of the architecture.
- Model Context Protocol, the open standard for how agents discover and call catalogued tools.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, the demand signal: task-specific AI agents projected to reach 40% of enterprise apps by 2026.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

03 GUARDRAILS GOVERNANCE

Human-in-the-Loop and reused governance: guardrails that scale.

More autonomy does not buy fewer controls. It demands more. An agent that can open a pull request, move money, or change a customer record needs explicit human checkpoints, placed by impact and reversibility rather than by intuition. This chapter maps the four Human-in-the-Loop quadrants, the three implementation patterns that cover most cases, and the governance an enterprise already owns in Azure and reuses instead of rebuilding.

More autonomy needs more guardrails: the four quadrants

More autonomy requires more guardrails, not fewer. The cost of one wrong autonomous action exceeds the savings of one hundred correct ones, so an agent that can act on regulated data, sign documents, move money, or change customer records needs explicit human checkpoints. Where you place those checkpoints is not a matter of taste. It follows from two properties of every action: business impact and reversibility. Impact is what breaks if the action is wrong. Reversibility is how cheaply you can undo it. Plot those two axes and each action lands in one of four quadrants, each with its own review mode.

Quadrant one, Automate, holds low-impact and highly reversible actions: suggesting a commit message, drafting an email reply, summarizing a meeting, searching documents. The agent runs these inside its own loop. You audit the log but add no checkpoint. Quadrant two, Review, holds high-impact and highly reversible actions: opening a pull request, scheduling a meeting, sending a draft to a client. A human approves before the commit or the send. Quadrant three, Confirm, holds low-impact and low-reversibility actions: deleting a temporary file, archiving a ticket, closing a conversation. A single confirmation step covers it. Quadrant four, Dual Control, holds high-impact and low-reversibility actions: signing a contract, transferring funds,

changing RBAC, deleting production data, filing taxes. Two humans approve, and the audit trail carries signed evidence. Most enterprise agents belong in quadrant two or three, not quadrant one.

> DEFINITION
>
> The four quadrants in one line: Automate for low impact and reversible, Review for high impact and reversible, Confirm for low impact and irreversible, Dual Control for high impact and irreversible. Place the checkpoint by impact and reversibility, never by intuition.

Three implementation patterns: approve-before, sample-and-review, and dual-control

Three implementation patterns cover most of what the quadrants demand. Approve-before has the agent propose an action and wait at a checkpoint before it acts. It fits pull requests, drafts, and recommendations, and it maps onto GitHub Copilot agent mode directly: the agent plans and edits, then pauses for a human to accept or reject before anything merges. Sample-and-review lets the agent act at high volume while a random sample of its output is audited and a KPI threshold triggers escalation. It fits tagging, classification, and routing. Dual-control requires two distinct people, an initiator and a separate approver, with a cryptographic signature on each side. It fits finance, legal, and production data, where the record of who approved what has to survive an audit.

The patterns also show how to scale review without becoming the bottleneck. Approve-before stays cheap when the agent's proposal is legible: a clear diff, a short rationale, and a link to the evidence it used. Encode that expectation once in .github/copilot-instructions.md so every run produces the same reviewable shape. Sample-and-review keeps quadrant one and the lighter parts of quadrant three fast: you accept that some low-impact actions ship without a human, and you defend that choice with the audit log and the sampling rate. Dual-control is non-negotiable for quadrant four, and it is where signed evidence earns its cost. Cost is itself a control. GitHub Copilot bills agent work in GitHub AI Credits, where one credit equals one US

cent, so a per-run credit budget is a quantitative guardrail you set and audit alongside the human checkpoints.

Reuse Azure governance instead of building parallel controls

The fastest way to make an agent program unsafe is to build a second set of controls next to the ones the enterprise already trusts. The controls an agent needs are the controls Azure already provides. Identity is Entra ID, the same directory that governs every other workload. Authorization is RBAC, scoped to what the task requires and nothing more. Content Safety screens prompts and responses against categories the organization has already defined. Purview classifies and tracks the data the agent touches, so a regulated record stays regulated when an agent reads it. Audit is per call: every tool invocation, every model request, every human approval lands in the same log the rest of the estate writes to. None of this is agent-specific. It is the standing governance of the platform, pointed at a new workload.

Tool access is the part most teams underestimate. An agent is only as safe as the tools it can call, so every tool is catalogued, versioned, and scoped through the Model Context Protocol before an agent may reach it. No agent talks to a random API. In GitHub Copilot, the MCP servers an agent may use are declared in configuration and constrained by scoped tokens, so the blast radius of any single tool is known in advance. Models are governed the same way: the ones an agent can select come from the GitHub Copilot picker, a fixed set that today includes Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro, and the GPT-5.x family, each with a known cost in GitHub AI Credits. Reusing identity, authorization, content safety, data governance, audit, and a scoped tool catalogue means the agent inherits the compliance posture the enterprise already runs.

Five principles for a governed agent program

Five principles hold a Microsoft agent program together, and each follows from the two sections above. Start narrow: one use case, one agent, one team, and generalize only after the value is proven, because a narrow scope is a small blast radius. Reuse governance: identity, RBAC, Content Safety, and Purview are the same controls as

the rest of the stack, not parallel ones, and every parallel control you avoid building is one you never have to audit twice. Catalogue every tool: an MCP server registry, scoped tokens, and audit per call mean no agent reaches a capability nobody signed off on. These three principles are about containment. They keep the set of things an agent can do small, known, and reviewable.

The last two principles are about time. Evaluate before deploy: keep a golden test set for each agent, run regression on every model swap, and hold a latency budget per loop, so a change to the model or the prompt cannot quietly change behavior a reviewer already approved. Swapping from one Copilot picker model to another is a model swap, and it passes the same evaluation gate as any other change. Plan the deprecation: version every prompt, log every change, and keep the rollback, because agents drift the way models do, and a program with no exit plan accumulates risk it cannot see. Guardrails are not a launch checklist you clear once. They are the standing shape of the program, refreshed as the agents change.

References

Primary sources for the platform, the guardrails, and the governance controls this chapter reuses.

- GitHub Copilot Documentation, agent mode, repository instruction files, and the model picker that this chapter governs.
- Azure AI Foundry Documentation, production governance for enterprise agents: identity, RBAC, and observability.
- What is Azure AI Content Safety?, the content filtering layer reused as an agent guardrail rather than rebuilt.
- Model Context Protocol, the open standard for cataloguing and scoping the tools an agent may call.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

04 EXECUTION PROOF

Vehicle Leasing in Production, and Beyond.

Two operational problems in vehicle leasing show what a multi-agent design looks like once it leaves the slide and runs in production. One is a traffic-fine deadline measured in days. The other is a tax document that blocks a month of invoices when one code is wrong. Both resolve into the same four moves, and the same four moves carry across four other verticals. This chapter is the execution proof.

Traffic-fine driver indication: four agents against a 15-day deadline

The first case ran in vehicle leasing, and it started with a legal clock. When a leased car gets a traffic fine, the law gives the leasing company 15 days to name the driver who was behind the wheel. Miss that window and the company absorbs the fine and a penalty on top of it. The manual process was fragile. The mailroom received a paper notice, an operator scanned it and retyped the fields into a spreadsheet, then emailed the customer who leased the car and waited for a reply. Deadlines slipped often enough to be a line item. Four specialized agents replaced that flow, and they run on Azure AI Foundry.

Each agent owns one job. An ingestion agent uses Document Intelligence to pull the plate number, infraction date, code, value, and deadline off the scanned notice. A lookup agent cross-references SAP and Salesforce to find the active lessee. A notification agent sends a portal link, an SMS, and an email. A tracking agent watches the clock: it escalates on day 7, sends a final ping on day 12, and routes the case to a paralegal on day 14. The notification template sits behind a Human-in-the-Loop checkpoint, so the operations team approves the wording before any message reaches a customer. Nothing here is exotic. The value comes from splitting one brittle manual chain into four agents that each do a narrow, testable thing.

One implementation detail matters for cost and repeatability. The four agents were specified and built with GitHub Copilot in agent mode. Every tool they call is declared as an MCP server, the leasing conventions live in .github/copilot-instructions.md, and each agent picks its model from the Copilot picker: Haiku 4.5 for cheap field extraction, Sonnet 5 for the lookup and notification logic, and Opus 4.8 where escalation timing has to reason over edge cases. Build cost is metered in GitHub AI Credits, where one credit is one cent. The same picker and the same instruction file carry over to the tax flow, which is why the second case took days to stand up rather than weeks.

NF-e tax compliance: three agents under dual-control filing

The second case has a sharper failure mode. Every contract closure, lease termination, and inter-branch vehicle transfer has to produce a valid Nota Fiscal Eletrônica, and the rules come from RICMS legislation that changes by state and by product category under the NCM table. One wrong CFOP code can block invoices for an entire month. Manual issuance holds at low volume and breaks at scale. Three agents carry the flow, triggered by an ERP event from SAP, Oracle, or TOTVS.

The first agent is a tax classifier. It uses Azure AI Search over the RICMS text and the NCM table to pick the correct CFOP, CST, and ICMS rate, so the classification is grounded in the actual legislation rather than inferred. The second agent issues the document against the SEFAZ NF-e 4.0 API and handles retries, contingency mode, and the cancellation path. The third agent runs a daily reconciliation across SPED, EFD-Contribuições, the in and out ledgers, and the ICMS-ST report. Monthly tax filing does not run unattended. It sits under a dual-control checkpoint: one person initiates, a second approves, and the CFO signs off with audited evidence. High impact and low reversibility earn the strictest guardrail in the framework.

The reusable pattern: ingest, classify, act, escalate

Look at both cases and the same skeleton shows up. An agent ingests a document or an event. An agent classifies it against rules the business already encodes,

whether that is a lessee record or a tax code. An agent acts, sending a notice or issuing a fiscal document. An agent escalates when a deadline approaches or a human sign-off is required. Ingest, classify, act, escalate. The pattern is deliberately boring, because boring is what survives an audit and a production incident review.

What changes between cases is not the shape, it is the guardrail. The traffic-fine flow reviews a template and escalates on a fixed calendar. The NF-e flow runs a daily reconciliation and gates the monthly filing behind dual control. Both decisions come straight from the Human-in-the-Loop framework: map each action by impact and reversibility, then place the checkpoint where a wrong autonomous action would cost more than the review. The pattern gives you the agents. The framework tells you where a human still has to stand.

> DEFINITION
>
> Ingest, classify, act, escalate. Four narrow agents, one Human-in-the-Loop checkpoint placed by impact and reversibility. Start with one use case and one team, prove the value in production, then reuse the skeleton across the next vertical.

Four verticals on the same pattern

The pattern does not stay in vehicle leasing. In finance, a document agent reads onboarding files, a risk-scoring agent matches names against sanctions lists, and an escalation agent routes to a compliance officer, with dual control on KYC and AML approval. In healthcare, a clinical-extraction agent pairs with a payer-rules RAG layer, and a physician reviews before any prior-authorization submission goes out. Authorization time drops from days to hours, and every decision is logged for audit. The same four moves, different rules, a different checkpoint.

Retail and manufacturing repeat it again. In retail, a triage agent classifies a return reason, a refund agent runs the policy check, and a logistics agent books the pickup, with the Human-in-the-Loop checkpoint reserved for exceptions above the SKU value threshold. In manufacturing, a telemetry agent watches IoT signals, a diagnostics agent proposes work orders, and a scheduling agent slots technicians,

with the plant manager approving any work order longer than four hours. Four verticals, one skeleton: ingest, classify, act, escalate, with the checkpoint placed where the cost of being wrong is highest. That is the proof. The pattern moved from a demo to production in leasing, and it transfers without a rewrite.

References

Primary sources for the platforms, tools, and governance controls behind the production cases.

- Azure AI Foundry documentation, the runtime the four leasing agents and the three NF-e agents run on.
- GitHub Copilot documentation, agent mode, .github/copilot-instructions.md, and the model picker used to build the agents.
- Model Context Protocol, the standard each agent tool is declared and scoped through.
- Azure AI Content Safety overview, the content and safety controls that back the Human-in-the-Loop checkpoints.
- Microsoft Copilot Studio documentation, the low-code path for the conversational surface of business-process agents.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

05 ADOPTION ROADMAP

From first pilot to production fleet: five phases, explicit exit criteria.

A working pilot is not a production program. The distance between one agent helping one team and a governed fleet serving the whole enterprise is measured in phases, not features. This chapter lays out the five-phase roadmap that runs over twelve months, the exit criteria that stop a pilot from shipping as a demo, and the cost discipline that keeps a fleet solvent once it runs on GitHub AI Credits.

The five phases: Discover, Pilot, Harden, Scale, Operate

The path from a first pilot to a production-grade agent fleet runs five phases over twelve months. Each phase changes both what you build and what you have to prove before you move on. Discover runs in month one. Pilot runs in months two and three. Harden runs in months four to six. Scale runs in months seven to nine. Operate begins in month ten and does not end. The sequence is deliberate, and you cannot skip a phase without moving its cost to a later one, where it is always larger and harder to pay.

Discover maps candidate use cases against value and effort, gets stakeholder sign-off, and selects one pilot candidate. Pilot deploys a single agent with a single team and runs a closed-loop evaluation before anyone else gets access. Harden implements RBAC and audit trails, sets SLA targets and cost caps, and wires Human-in-the-Loop checkpoints into every action that touches regulated data or irreversible state. These first three phases are how an idea becomes something a review board is willing to approve for real users.

Scale builds the tool registry, introduces multi-agent orchestration, and opens self-service onboarding so new teams can bring their own use cases without a platform engineer in the loop for every one. Operate establishes the disciplines that keep a fleet healthy for years: FinOps, drift monitoring, model-swap drills, and quarterly

reviews against the value targets that were written down back in Discover. Scale grows the surface area of the program. Operate is what keeps that surface honest.

Explicit exit criteria per phase

Each phase has an exit gate, and the gate is a set of conditions, not a date on the calendar. Discover exits when one use case is chosen and its value target is written down in a number someone will be held to. Pilot exits when the closed-loop evaluation clears its golden test set and the sponsoring team actually wants to keep using the agent. Harden exits when every high-impact, low-reversibility action sits behind a dual-control checkpoint and every call the agent makes is audited.

The two failures I see most often are skipping the gate after Pilot and skipping the gate after Scale. Piloting without hardening produces a demo: it works in the room, and it falls over the first time it meets real permissions, real audit requirements, and a real invoice. Scaling without operating produces debt: more agents, more teams, and more spend, with no one watching for drift or reading the monthly bill. The exit criteria exist to make both of those failures visible early, before they compound across the fleet.

> DEFINITION
>
> Each phase has explicit exit criteria. Piloting without hardening produces a demo. Scaling without operating produces debt.

Operate for the long run

Operate is where most programs quietly fail, because the work is unglamorous and it never ends. Four disciplines carry it. FinOps tracks spend per agent, per team, and per loop, and holds each against the cost cap set in Harden. Drift monitoring watches for the slow decay that hits agents the same way it hits models: a prompt that worked six months ago starts to miss, and the golden test set catches it before your users do. Model-swap drills rehearse changing the model underneath an agent without changing how the agent behaves.

Model-swap drills matter because the model behind an agent is never fixed for long. In GitHub Copilot, the model picker exposes Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro, and the GPT-5.x family. A new entry in the picker is a chance to cut cost or raise quality, but only if you can swap it in and re-run the golden test set to prove the agent still behaves. Rehearse that swap on a schedule, not during an incident. Quarterly reviews then close the loop: measure each agent against its original value target, and retire the ones that no longer earn their place.

Cost in practice: spend per loop

Cost is a first-class metric here, not a year-end surprise. On GitHub Copilot, agent usage is billed in GitHub AI Credits, and one credit is worth one cent. Since June 2026 that is the unit every FinOps report converts back to. Tracking spend per loop means attributing each plan-act-observe cycle to a credit cost, so a team can see which agents are cheap, which are expensive, and which ones are expensive for a good reason.

The discipline compounds across the whole roadmap. Harden sets the cost cap. Scale multiplies the number of loops running every day. Operate reads the meter and reports it. An agent that costs a handful of credits per loop is fine at pilot volume and a real line item at fleet volume, so the latency and token budget you set per loop in Harden is the same budget that keeps the fleet solvent in Operate. Cost per loop, measured in credits, is the single number that connects the design of one agent to the bill for the entire program.

References

Primary sources for the platforms, the billing model, and the governance controls this roadmap depends on.

- GitHub Copilot Documentation, agent mode and the GitHub AI Credits billing model that meters spend per loop.
- Azure AI Foundry Documentation, the model catalog and production observability behind model-swap drills and drift monitoring.

- Microsoft Copilot Studio Documentation, the low-code path for the self-service onboarding opened in the Scale phase.
- Model Context Protocol, the open standard behind the catalogued, scoped tool registry built in Scale.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, the enterprise adoption trajectory that frames the twelve-month roadmap.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps