CONSTRUINDO AGENTES DE IA ENTERPRISE NA MICROSOFT · PLAYBOOK

Um agente percebe, decide e age, delimitado por política.

Agentes de IA enterprise não são chatbots. Eles percebem, decidem e agem, com autonomia delimitada por política. Este playbook mapeia as três plataformas Microsoft, Copilot Studio, GitHub Copilot e Azure AI Foundry, os quatro tipos de agente e a arquitetura de seis camadas, os guardrails de Human-in-the-Loop que escalam, dois casos em produção e um roteiro de adoção de doze meses, tudo visto pelo GitHub Copilot.

AUTORA

Paula Silva

CARGO

AI-Native Software Engineer

CONTATO

in/paulanunes

TEMPO DE LEITURA

~ 45 minutos, fim a fim

3
PLATAFORMAS MICROSOFT

6
CAPÍTULOS

4
TIPOS DE AGENTE

5
FASES DE ADOÇÃO

CAPÍTULOS

Chapter 00

O que um agente é, e o que não é

A definição de trabalho, perceber, decidir, agir, delimitado por política, chatbot versus agente, os cinco componentes que todo agente compartilha e o stack de decisão de quatro camadas.

Fundamentos →

Chapter 01

Três plataformas, um framework

Como escolher entre Copilot Studio, GitHub Copilot e Azure AI Foundry por público, nível de skill, tempo até o piloto e modelo de governança.

Plataformas →

Chapter 02

Tipos e a arquitetura de seis camadas

Os quatro tipos de agente enterprise combinados com o trabalho, as seis camadas da governança até a superfície, ferramentas MCP, escolha de modelo e RAG do jeito certo.

Arquitetura →

Chapter 03

Governança Human-in-the-Loop

Os quatro quadrantes de HITL por impacto e reversibilidade, três padrões de implementação, e como reutilizar a governança do Azure em vez de controles paralelos.

Governança →

Chapter 04

Casos em produção

Locação de veículos em produção: quatro agentes contra um prazo de multa de 15 dias, três agentes para conformidade fiscal de NF-e, e o padrão ingerir, classificar, agir, escalonar em quatro verticais.

Caso real →

Chapter 05

Roteiro de adoção

O roteiro de cinco fases ao longo de doze meses, da Descoberta à Operação, critérios de saída explícitos por fase, e custo medido por loop em GitHub AI Credits.

Roteiro →

DICAS DE LEITURA

ATALHOS DE TECLADO

Aperte `/` para buscar. Use `j` e `k` para mover entre capítulos. Aperte `t` para alternar tema. Aperte `g` para ir ao topo.

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 ANATOMIA DO AGENTE

O que um agente de IA é, e o que não é: perceber, decidir, agir, delimitado por política.

Um agente de IA percebe, decide e age, e sua autonomia é delimitada por política. Essa única frase separa os agentes de hoje dos chatbots de ontem, e prepara tudo o que vem a seguir. Este capítulo dá a definição de trabalho, contrasta um chatbot stateless com um agente orientado a metas, nomeia os cinco componentes que todo agente compartilha e os posiciona em um stack de decisão de quatro camadas, tudo visto pelo GitHub Copilot.

Perceber, decidir, agir: a definição de trabalho, delimitada por política

Um agente de IA é um software que percebe, decide e age. Sua autonomia é delimitada por política. Essa definição é curta de propósito, e cada palavra carrega peso. Perceber significa que o agente lê sua entrada: um prompt, um arquivo, um repositório, um evento, o resultado da sua última ação. Decidir significa que ele planeja o próximo passo em direção a uma meta, em vez de devolver uma única resposta pronta. Agir significa que ele muda algo no mundo: edita um arquivo, chama uma API, executa um comando, abre um pull request. A linha entre o que existia antes e o que as empresas implantam hoje passa exatamente por esses três verbos.

A quarta cláusula é a que os times esquecem. A autonomia é delimitada por política, e o limite faz parte da definição, não é um controle que você acopla depois. O GitHub Copilot em agent mode torna isso concreto. Ele lê seu repositório, planeja uma mudança, edita arquivos, executa o build e abre um pull request, e faz tudo isso dentro dos limites que você escreve em .github/copilot-instructions.md. Esse arquivo de instruções é onde a política vive: o que o agente pode tocar, quais convenções

deve seguir, quando precisa parar e perguntar. Um agente sem política não é mais capaz. É apenas um agente sem supervisão.

> DEFINIÇÃO
>
> Um agente de IA percebe, decide e age, com autonomia delimitada por política. A plataforma que você escolhe define o limite. A arquitetura que você entrega conquista a confiança.

Chatbot versus agente: stateless e reativo versus orientado a metas, stateful e usando ferramentas

O chatbot de ontem era stateless e reativo. Recebia uma pergunta e devolvia uma única resposta. Não guardava memória entre turnos, não conseguia chamar ferramentas e não tinha noção de perseguir uma meta. Pergunte duas vezes e ele começa do zero a cada vez. Esse design servia bem para perguntas frequentes e fluxos de suporte roteirizados, e ainda serve para isso. Ele não é um agente, e chamá-lo assim cria a expectativa errada.

O agente de hoje é orientado a metas, stateful e capaz de usar ferramentas. Ele recebe um objetivo, planeja um caminho para alcançá-lo, chama ferramentas externas, observa os resultados e itera até a meta ser atingida, ou escalona para um humano quando não consegue prosseguir com segurança. A diferença não é o modelo. O mesmo modelo no Copilot picker pode conduzir uma resposta de uma vez só ou um agente de múltiplos passos. A diferença é o loop ao redor dele e o estado que ele mantém entre os passos. Quando o GitHub Copilot passa de uma resposta de chat para o agent mode, o modelo não muda; o loop, a memória e o acesso às ferramentas mudam.

Os cinco componentes que todo agente compartilha: Brain, Memory, Tools, Policy, Orchestration

Todo agente, independentemente da plataforma, é construído a partir de cinco componentes. O Brain é o modelo que raciocina sobre a entrada e planeja o próximo passo, escolhido por tarefa. No GitHub Copilot o Brain é o modelo que você seleciona no Copilot picker: Claude Opus 4.8 para raciocínio profundo, Claude Fable 5 ou Sonnet 5 para trabalho equilibrado, Haiku 4.5 para turnos rápidos e baratos, Gemini 3.1 Pro, ou um modelo GPT-5.x. Você escolhe por tarefa, não uma vez para o programa inteiro. A Memory é o histórico de conversa, um bloco de rascunho de trabalho, um vector store para recuperação e memória episódica entre execuções. A Memory é delimitada por política: um agente lembra apenas o que tem permissão para lembrar.

Tools, Policy e o loop que amarra tudo

As Tools são as funções, APIs, consultas a bancos de dados, execução de código e acionadores de workflow que o agente pode chamar. Elas são catalogadas e com escopo definido pelo Model Context Protocol: cada servidor MCP é um conjunto versionado e nomeado de ferramentas com um limite definido, para que o agente nunca converse com um endpoint aleatório. A Policy é RBAC, filtros de conteúdo, limites de taxa, checkpoints de Human-in-the-Loop e trilhas de auditoria, e é inegociável; no GitHub Copilot o arquivo .github/copilot-instructions.md e as regras da sua organização a carregam, e o Azure AI Content Safety aplica os filtros de conteúdo. A Orchestration é o loop plan-act-observe, a coordenação multi-agente, as retentativas e os timeouts, e é responsabilidade da plataforma. O Brain raciocina. A Memory persiste. As Tools agem no mundo. A Policy delimita. A Orchestration conduz o loop.

O stack de decisão como modelo mental: anatomia, depois plataforma, depois tipo, depois Human-in-the-Loop

Mantenha essas partes em um stack de decisão de quatro camadas, porque cada camada restringe a de baixo. A Camada I é a Anatomia: perceber, decidir, agir, e os cinco componentes. A Camada II é a Plataforma: Copilot Studio para citizen developers, GitHub Copilot Agents para times de engenharia, Azure AI Foundry para produção regulada. A Camada III é o Tipo: desenvolvimento, produtividade, processo de negócio, ou dados e analytics, casado com a família de tarefa e a persona dona. A Camada IV é Human-in-the-Loop: quatro quadrantes definidos por impacto e reversibilidade, da autonomia total à aprovação por duas pessoas.

Leia o stack de baixo para cima, e construa nessa ordem. Acerte a anatomia primeiro, porque uma escolha de plataforma não conserta uma definição quebrada. Depois escolha a plataforma, depois case o tipo de agente com o trabalho, depois defina o limite de Human-in-the-Loop que o risco da ação exige. Pule uma camada e o agente chega à produção sem supervisão. A pressão para pular é real: o Gartner projeta que 40% das aplicações corporativas terão agentes de IA de tarefa específica até 2026, ante menos de 5% em 2025. A anatomia é a camada que você não pode pular. Todo o resto deste playbook se apoia nela.

Referências

Fontes primárias para a definição de trabalho, os cinco componentes e as referências de plataforma e protocolo usadas ao longo deste capítulo.

- GitHub Copilot Documentation, a plataforma de referência para agent mode, agentes customizados e o arquivo copilot-instructions usado ao longo deste playbook.
- Model Context Protocol Specification, o padrão aberto que cataloga e define o escopo das ferramentas que um agente pode chamar.
- Microsoft Copilot Studio Documentation, a plataforma low-code para usuários de negócio e citizen developers.

- Azure AI Foundry Documentation, a plataforma para orquestração multi-agente e agentes de nível produção.
- What is Azure AI Content Safety?, os filtros de conteúdo que aplicam parte da camada de política de um agente.
- Gartner, task-specific AI agents in 40% of enterprise apps by 2026, a projeção de adoção: 40% das aplicações corporativas terão agentes de IA de tarefa específica até 2026, ante menos de 5% em 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 ESCOLHA DE PLATAFORMA

Três plataformas, um framework de decisão: Copilot Studio, GitHub Copilot e Azure AI Foundry.

A Microsoft entrega três plataformas para construir agentes de IA: Copilot Studio, GitHub Copilot e Azure AI Foundry. Elas são complementares, não concorrentes, e a maioria das empresas usa as três. A pergunta difícil nunca é qual plataforma vence. É qual caso de uso vai para qual plataforma. Este capítulo entrega um framework de decisão construído sobre quatro eixos: público, nível de skill, tempo até o piloto e modelo de governança.

Três plataformas, três públicos

O Copilot Studio é voltado para usuários de negócio e desenvolvedores cidadãos. Oferece autoria de tópicos por drag-and-drop, conectores nativos para Microsoft 365, Dataverse e Power Platform, e um ciclo de deploy medido em horas a dias. Alguém que nunca abriu uma IDE consegue publicar um agente funcional, e esse é o ponto. O Copilot Studio se encaixa quando quem constrói está mais perto do processo de negócio do que da base de código, e quando o agente vive dentro do Microsoft 365 e não dentro de um repositório. Autoatendimento de RH, automação de helpdesk de TI e bots de atendimento ao cliente são a casa natural dele.

O GitHub Copilot é voltado para engenheiros de software que trabalham dentro da IDE e do repositório. O agent mode lê o código, planeja uma mudança, edita arquivos, executa comandos e abre um pull request, tudo dentro do fluxo do desenvolvedor. O Azure AI Foundry é voltado para engenheiros de ML e times de plataforma. Ele oferece o SDK completo, um catálogo de modelos, orquestração multi-agente, pipelines de fine-tuning e observabilidade de nível produção. O tempo até o piloto no Foundry corre em semanas, mas o teto é muito mais alto. As três plataformas são complementares. O Copilot Studio é dono da superfície

conversacional, o GitHub Copilot é dono do loop de engenharia, e o Azure AI Foundry é dono dos agentes verticais que carregam SLAs rigorosos.

Escolhendo por público, skill, tempo até o piloto e governança

Quatro eixos decidem para onde vai um caso de uso. O público vem primeiro: negócio e desenvolvedores cidadãos apontam para o Copilot Studio, engenheiros de software apontam para o GitHub Copilot, e times de ML ou plataforma apontam para o Azure AI Foundry. O nível de skill vem em segundo: o Copilot Studio é low-code e no-code, o GitHub Copilot é code-first e nativo na IDE, e o Azure AI Foundry é pro-code com SDK completo. Leia esses dois eixos juntos e a maioria das decisões se resolve sozinha. Quem constrói e onde se constrói já dizem qual plataforma abrir.

O tempo até o piloto é o terceiro eixo. O Copilot Studio entrega em horas a dias, o GitHub Copilot em dias, e o Azure AI Foundry em semanas. A governança é o quarto, e os times a subestimam. O Copilot Studio herda a administração do Power Platform, o GitHub Copilot herda as políticas do GitHub, e o Azure AI Foundry herda o Azure RBAC e o Content Safety. Escolha a plataforma cuja governança nativa já corresponde ao risco do caso de uso. Adaptar controles depois do piloto custa mais do que escolher certo no primeiro dia. O Gartner prevê que 40% dos aplicativos corporativos terão agentes de IA específicos por tarefa até 2026, contra menos de 5% em 2025, então o volume dessas decisões está prestes a crescer.

O agent mode do GitHub Copilot no repositório

O agent mode é onde o GitHub Copilot deixa de sugerir linhas e passa a fazer o trabalho. Ele lê o repositório, planeja uma mudança, edita arquivos, executa comandos e abre um pull request para revisão. O comportamento é moldado por um arquivo: .github/copilot-instructions.md. Esse arquivo carrega as convenções que a base de código já codifica, tratamento de exceções, formato de logging, organização de testes e nomenclatura, para que cada sessão de agent mode as herde em vez de adivinhar. Escreva uma vez e cada tarefa começa fundamentada. As ferramentas MCP estendem o agente para além do repositório. Cada servidor do Model Context Protocol é catalogado, versionado e com escopo definido, dando ao

agente acesso a sistemas de build, rastreadores de tickets e serviços internos sem nunca conversar com uma API aleatória.

O seletor de modelos do Copilot transforma a escolha de modelo em uma decisão por tarefa. Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro e GPT-5.x ficam todos no mesmo seletor, e você direciona a tarefa para o modelo que combina com seu custo e sua dificuldade. Desde 1 de junho de 2026, o uso de modelos premium é cobrado como GitHub AI Credits, onde 1 crédito equivale a US$ 0,01. Cada sessão de agent mode e cada tarefa em segundo plano do Coding Agent é uma linha de fatura medida. Trate esses créditos como gasto de infraestrutura, faça orçamento por time, e a história de FinOps continua honesta.

> DEFINIÇÃO
>
> O agent mode se fundamenta no .github/copilot-instructions.md e alcança o mundo externo por servidores MCP com escopo definido. A escolha de modelo é uma decisão por tarefa no seletor do Copilot, e cada chamada premium é medida como GitHub AI Credits a 1 crédito = US$ 0,01.

Onde cada caso de uso vai parar

Mapeie os quatro eixos sobre o trabalho real e as fronteiras ficam nítidas. Autoatendimento de RH e automação de helpdesk de TI vão para o Copilot Studio: quem constrói é um dono de negócio, os conectores para o Microsoft 365 são nativos, e o piloto entrega em horas a dias. Geração de código, refatoração, automação de PRs e geração de testes vão para o agent mode do GitHub Copilot: o trabalho vive no repositório, as convenções vivem no .github/copilot-instructions.md, e o pull request é o checkpoint. Nenhum desses casos de uso precisa de um SDK completo, e forçá-los para dentro de um desperdiça semanas.

Agentes verticais regulados vão para o Azure AI Foundry. Um fluxo de KYC e AML em finanças, um fluxo de autorização prévia em saúde, ou um fluxo fiscal de NF-e com prazo legal de 15 dias exigem orquestração multi-agente, Azure RBAC, Content Safety e Human-in-the-Loop com controle duplo nas ações irreversíveis. Esse é o teto que só o Foundry oferece, e as semanas de tempo até o piloto compram os

SLAs que a vertical exige. A regra que vale para as três plataformas é a mesma: comece estreito. Um caso de uso, um agente, um time, e generalize apenas depois que o valor for comprovado.

Um parque, um framework

A maioria das empresas não vai escolher uma única plataforma. Vai rodar Copilot Studio, GitHub Copilot e Azure AI Foundry lado a lado, cada um carregando os casos de uso que combinam com ele. O framework é o que mantém esse parque coerente: os mesmos quatro eixos, aplicados por caso de uso, para que cada agente vá parar na plataforma cujo público, nível de skill, tempo até o piloto e governança já correspondem a ele.

Referências

Fontes primárias para as três plataformas, seu modelo de governança e os padrões de integração de agentes deste capítulo.

- Microsoft Copilot Studio documentation, a plataforma low-code para usuários de negócio e desenvolvedores cidadãos.
- GitHub Copilot documentation, a referência para o agent mode, o .github/copilot-instructions.md, as ferramentas MCP e o seletor de modelos.
- Azure AI Foundry documentation, a plataforma pro-code para orquestração multi-agente e agentes verticais regulados.
- What is Azure AI Content Safety?, o controle de governança que o Azure AI Foundry herda para cargas reguladas.
- Model Context Protocol, o padrão aberto que dá escopo e cataloga as ferramentas que um agente pode chamar.
- Gartner, agentes de IA específicos por tarefa em 40% dos aplicativos corporativos até 2026, a projeção de mercado por trás do volume de decisões de plataforma que vem pela frente.

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 MÉTODO E ARQUITETURA

Tipos de agente e a arquitetura de seis camadas: combine o trabalho, depois empilhe a plataforma.

Um programa de agentes enterprise tem duas decisões de design para acertar antes de qualquer código ir para produção. A primeira é o tipo de agente, que decorre do trabalho e da persona que é dona dele. A segunda é a arquitetura, seis camadas empilhadas da governança até a superfície onde o usuário encontra o agente. Este capítulo define os quatro tipos, percorre as seis camadas de baixo para cima e mostra onde ferramentas, modelos e contexto se encaixam.

Quatro tipos de agente, combinados com o trabalho

Quatro tipos de agente cobrem a maior parte da demanda enterprise, e cada um se mapeia claramente em uma plataforma. Agentes de desenvolvimento vivem no repositório e na IDE. O GitHub Copilot em agent mode é o ambiente natural, com instruções de repositório em .github/copilot-instructions.md e ferramentas MCP conectadas a sistemas de build e trackers de tickets. Tarefas típicas são correção de bugs, refatoração e geração de testes. Agentes de produtividade operam dentro do Microsoft 365. O Copilot Studio cuida da autoria de tópicos customizados, e o Microsoft 365 Copilot Agents entrega experiências embarcadas no Teams, Outlook e SharePoint. Tarefas típicas são notas de reunião, rascunhos de e-mail e busca em documentos.

Os outros dois tipos carregam mais risco. Agentes de processos de negócio orquestram entre sistemas. O Copilot Studio é dono da camada conversacional, e o Azure AI Foundry assume quando o fluxo exige SLAs rigorosos, coordenação multi-agente ou fine-tuning. Tarefas típicas são onboarding de colaboradores, fluxos de aprovação e processamento de sinistros. Agentes de dados e analytics fundamentam suas respostas em dados corporativos no Fabric, Synapse ou em um

data warehouse próprio. Exigem engenharia de prompt rigorosa, um harness de avaliação minucioso e aprovação humana antes de publicar qualquer resultado. Tarefas típicas são linguagem natural para SQL e resumos de insights. O tipo não é um rótulo. Ele define a governança que você deve ao agente.

As seis camadas, de baixo para cima

Programas de agentes de produção são construídos em seis camadas, e cada camada restringe a que está acima dela. A base é Plataforma e Governança: identidade, RBAC, Content Safety, auditoria, observabilidade e FinOps. São os mesmos controles que já operam o restante do Azure, não uma stack paralela. Acima dela ficam os Modelos: um catálogo de modelos escolhidos por tarefa, com fine-tuning apenas onde o retorno justifica o custo. Acima dos Modelos vem Contexto e RAG: um vector store, recuperação semântica e acesso com escopo por identidade aos dados corporativos. Essas três camadas decidem o que o agente sabe antes de agir.

Ferramentas, orquestração e a superfície

As três camadas de cima decidem o que o agente faz. Ferramentas e MCP é o registro de servidores MCP e plugins catalogados, versionados e com escopo definido, que deixam o agente agir no mundo. Orquestração é o runtime: o loop planejar, agir, observar, coordenação multi-agente, retentativas, checkpoints HITL e avaliação. Superfície é onde os usuários encontram o agente, no Teams, Outlook, na IDE ou em um app customizado. Pule uma camada e a lacuna não desaparece. Ela vai para produção, sem supervisão. Comece estreito, prove um caso de uso nas seis camadas, depois generalize.

> DEFINIÇÃO
>
> Seis camadas, de baixo para cima: Plataforma e Governança, Modelos, Contexto e RAG, Ferramentas e MCP, Orquestração e Superfície. Cada camada restringe a que está acima. Pule uma e o agente chega à produção sem o controle que aquela camada deveria impor.

Catalogando ferramentas via MCP

Ferramentas são onde o agente para de conversar e começa a agir, então esta é a camada que precisa da disciplina mais rígida. Cada ferramenta que um agente pode chamar é uma função, uma API, uma consulta a banco de dados ou um acionamento de workflow, e cada uma é exposta pelo Model Context Protocol. O MCP é o padrão aberto de como um agente descobre e chama ferramentas, e transforma um conjunto ad hoc de integrações em um catálogo governado. A regra é simples. Nenhum agente conversa com uma API aleatória. Cada ferramenta é registrada em um registro de servidores MCP, versionada e com escopo para a menor permissão que resolve a tarefa.

O catálogo carrega três obrigações. Registrar cada servidor para que um time de plataforma veja o que existe e aposente o que fica obsoleto. Dar escopo a cada token para que o agente tenha apenas o acesso que sua tarefa exige, nunca os direitos completos do usuário. Auditar cada chamada para que um revisor consiga reconstruir o que o agente fez e por quê. No GitHub Copilot, os mesmos servidores MCP sustentam o agent mode na IDE, o que significa que um agente de desenvolvimento e um agente de processos de negócio tiram ferramentas de uma única fonte governada, e não de duas divergentes. Uma ferramenta catalogada é revisável. Uma não catalogada é um passivo esperando um incidente.

Escolhendo o modelo por tarefa no seletor do Copilot

O modelo é uma escolha, não um padrão. No GitHub Copilot o seletor expõe vários modelos, e o certo depende da tarefa à sua frente. Claude Fable 5, Opus 4.8, Sonnet 5 e Haiku 4.5 ficam ao lado do Gemini 3.1 Pro e da família GPT-5.x. Uma refatoração curta com orçamento de latência apertado não precisa do maior modelo de raciocínio. Uma migração multi-arquivo com dependências profundas precisa. Combine o modelo com a dificuldade da tarefa, e reavalie a combinação quando a tarefa ou o modelo mudarem.

Custo faz parte da escolha. O GitHub Copilot cobra o uso de modelos em GitHub AI Credits, onde um crédito equivale a um centavo de dólar, em vigor desde 1 de junho de 2026. Um modelo maior gasta mais créditos por requisição, então o seletor

também é uma decisão de orçamento, não só de capacidade. Trate cada troca de modelo como uma mudança que precisa de prova. Mantenha um golden test set por agente, rode regressão a cada troca e mantenha um orçamento de latência por loop. Um modelo que passa no eval ganha o tráfego. Um que não passa fica de fora, seja qual for sua reputação.

Contexto e RAG do jeito certo

A camada de Contexto e RAG decide o que o agente sabe antes de raciocinar. Bem feita, ela fundamenta as respostas em dados corporativos e corta as alucinações que surgem quando um modelo chuta fatos que nunca recebeu. Três peças a fazem funcionar. Um vector store guarda embeddings do corpus corporativo, de documentos a código e tickets. A recuperação semântica traz as passagens que realmente importam para o pedido, não as que apenas compartilham palavras-chave. E o acesso com escopo por identidade garante que o agente recupere só o que o usuário que o invoca tem permissão de ver.

Essa terceira peça é a que os times pulam, e é a que mais importa em um parque regulado. Recuperação com escopo por identidade significa que o vector store honra o mesmo RBAC que governa os sistemas de origem. Um agente agindo por um engenheiro de suporte não pode trazer à tona um documento que o engenheiro jamais conseguiria abrir diretamente. Ligue a recuperação à camada de identidade da base de Plataforma e Governança, não por fora dela. Um agente de dados e analytics que se fundamenta no Fabric ou no Synapse ainda deve aprovação humana antes de publicar, porque fundamentar reduz a taxa de alucinação, mas não a zera.

Referências

Fontes primárias para as plataformas, o protocolo de ferramentas e os controles de governança que este capítulo usa.

- Microsoft Copilot Studio documentation, a plataforma para agentes de produtividade e de processos de negócio construídos por citizen developers.

- GitHub Copilot documentation, agent mode, instruções de repositório e o seletor de modelos para agentes de desenvolvimento.
- Azure AI Foundry documentation, o catálogo de modelos e o runtime multi-agente para agentes verticais com SLAs rigorosos.
- What is Azure AI Content Safety?, um controle na camada de Plataforma e Governança na base da arquitetura.
- Model Context Protocol, o padrão aberto de como os agentes descobrem e chamam ferramentas catalogadas.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, o sinal de demanda: agentes de IA task-specific projetados para chegar a 40% dos apps enterprise até 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 E GOVERNANÇA

Human-in-the-Loop e governança reutilizada: guardrails que escalam.

Mais autonomia não compra menos controles. Exige mais. Um agente que pode abrir um pull request, movimentar dinheiro ou alterar um registro de cliente precisa de checkpoints humanos explícitos, posicionados por impacto e reversibilidade, não por intuição. Este capítulo mapeia os quatro quadrantes de Human-in-the-Loop, os três padrões de implementação que cobrem a maioria dos casos, e a governança que a empresa já tem no Azure e reutiliza em vez de reconstruir.

Mais autonomia exige mais guardrails: os quatro quadrantes

Mais autonomia exige mais guardrails, não menos. O custo de uma ação autônoma errada supera a economia de cem ações corretas, então um agente que pode agir sobre dados regulados, assinar documentos, movimentar dinheiro ou alterar registros de clientes precisa de checkpoints humanos explícitos. Onde você posiciona esses checkpoints não é questão de gosto. Decorre de duas propriedades de cada ação: impacto de negócio e reversibilidade. Impacto é o que quebra se a ação estiver errada. Reversibilidade é quão barato é desfazê-la. Cruze esses dois eixos e cada ação cai em um de quatro quadrantes, cada um com seu próprio modo de revisão.

O quadrante um, Automatizar, reúne ações de baixo impacto e alta reversibilidade: sugerir uma mensagem de commit, redigir uma resposta de e-mail, resumir uma reunião, buscar documentos. O agente executa essas ações dentro do próprio loop. Você audita o log, mas não adiciona checkpoint. O quadrante dois, Revisar, reúne ações de alto impacto e alta reversibilidade: abrir um pull request, agendar uma reunião, enviar um rascunho a um cliente. Um humano aprova antes do commit ou do envio. O quadrante três, Confirmar, reúne ações de baixo impacto e baixa

reversibilidade: deletar um arquivo temporário, arquivar um ticket, encerrar uma conversa. Um único passo de confirmação resolve. O quadrante quatro, Controle Duplo, reúne ações de alto impacto e baixa reversibilidade: assinar um contrato, transferir fundos, alterar RBAC, deletar dados de produção, enviar declaração fiscal. Dois humanos aprovam, e a trilha de auditoria carrega evidência assinada. A maioria dos agentes corporativos pertence ao quadrante dois ou três, não ao um.

> DEFINIÇÃO
>
> Os quatro quadrantes em uma linha: Automatizar para baixo impacto e reversível, Revisar para alto impacto e reversível, Confirmar para baixo impacto e irreversível, Controle Duplo para alto impacto e irreversível. Posicione o checkpoint por impacto e reversibilidade, nunca por intuição.

Três padrões de implementação: aprovar antes, amostrar e revisar, e controle duplo

Três padrões de implementação cobrem a maior parte do que os quadrantes exigem. Aprovar antes faz o agente propor uma ação e aguardar em um checkpoint antes de agir. Serve para pull requests, rascunhos e recomendações, e se mapeia diretamente no agent mode do GitHub Copilot: o agente planeja e edita, depois pausa para um humano aceitar ou rejeitar antes de qualquer merge. Amostrar e revisar deixa o agente agir em alto volume enquanto uma amostra aleatória de sua saída é auditada e um limiar de KPI dispara a escalação. Serve para tagging, classificação e roteamento. Controle duplo exige duas pessoas distintas, um iniciador e um aprovador separado, com uma assinatura criptográfica de cada lado. Serve para finanças, jurídico e dados de produção, onde o registro de quem aprovou o quê precisa sobreviver a uma auditoria.

Os padrões também mostram como escalar a revisão sem virar gargalo. Aprovar antes fica barato quando a proposta do agente é legível: um diff claro, uma justificativa curta e um link para a evidência usada. Codifique essa expectativa uma vez em .github/copilot-instructions.md para que cada execução produza o mesmo formato revisável. Amostrar e revisar mantém rápidos o quadrante um e as partes mais leves do quadrante três: você aceita que algumas ações de baixo impacto

seguem sem um humano, e defende essa escolha com o log de auditoria e a taxa de amostragem. Controle duplo é inegociável para o quadrante quatro, e é onde a evidência assinada justifica seu custo. O custo é, ele próprio, um controle. O GitHub Copilot cobra o trabalho de agente em GitHub AI Credits, em que um crédito equivale a um centavo de dólar, então um orçamento de créditos por execução é um guardrail quantitativo que você define e audita junto com os checkpoints humanos.

Reutilize a governança do Azure em vez de criar controles paralelos

O jeito mais rápido de tornar um programa de agentes inseguro é construir um segundo conjunto de controles ao lado dos que a empresa já confia. Os controles de que um agente precisa são os controles que o Azure já oferece. Identidade é o Entra ID, o mesmo diretório que governa qualquer outro workload. Autorização é RBAC, com escopo no que a tarefa exige e nada além. O Content Safety filtra prompts e respostas contra as categorias que a organização já definiu. O Purview classifica e rastreia os dados que o agente toca, para que um registro regulado siga regulado quando um agente o lê. A auditoria é por chamada: cada invocação de ferramenta, cada requisição de modelo, cada aprovação humana cai no mesmo log que o restante do parque escreve. Nada disso é específico de agente. É a governança permanente da plataforma, apontada para um novo workload.

O acesso a ferramentas é a parte que a maioria dos times subestima. Um agente é tão seguro quanto as ferramentas que pode chamar, então cada ferramenta é catalogada, versionada e com escopo definido pelo Model Context Protocol antes que um agente possa alcançá-la. Nenhum agente conversa com uma API aleatória. No GitHub Copilot, os servidores MCP que um agente pode usar são declarados em configuração e restritos por tokens com escopo, de modo que o raio de impacto de qualquer ferramenta é conhecido de antemão. Os modelos são governados do mesmo jeito: os que um agente pode selecionar vêm do seletor do GitHub Copilot, um conjunto fixo que hoje inclui Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro e a família GPT-5.x, cada um com um custo conhecido em GitHub AI Credits. Reutilizar identidade, autorização, content safety, governança de dados, auditoria e um catálogo de ferramentas com escopo faz o agente herdar a postura de conformidade que a empresa já opera.

Cinco princípios para um programa de agentes governado

Cinco princípios sustentam um programa de agentes da Microsoft, e cada um decorre das duas seções acima. Comece estreito: um caso de uso, um agente, um time, e generalize apenas depois que o valor for comprovado, porque escopo estreito é raio de impacto pequeno. Reutilize a governança: identidade, RBAC, Content Safety e Purview são os mesmos controles do restante da stack, não paralelos, e cada controle paralelo que você deixa de construir é um que você nunca precisa auditar duas vezes. Catalogue cada ferramenta: um registro de servidores MCP, tokens com escopo e auditoria por chamada garantem que nenhum agente alcance uma capacidade que ninguém aprovou. Esses três princípios tratam de contenção. Mantêm o conjunto de coisas que um agente pode fazer pequeno, conhecido e revisável.

Os dois últimos princípios tratam de tempo. Avalie antes de fazer deploy: mantenha um golden test set por agente, rode regressão a cada troca de modelo e respeite um orçamento de latência por loop, para que uma mudança no modelo ou no prompt não altere em silêncio um comportamento que um revisor já aprovou. Trocar de um modelo do seletor do Copilot para outro é uma troca de modelo, e passa pelo mesmo portão de avaliação que qualquer outra mudança. Planeje a descontinuação: versione cada prompt, registre cada mudança e mantenha o rollback, porque agentes derivam como modelos derivam, e um programa sem plano de saída acumula risco que não consegue enxergar. Guardrails não são um checklist de lançamento que você cumpre uma vez. São o formato permanente do programa, atualizado conforme os agentes mudam.

Referências

Fontes primárias para a plataforma, os guardrails e os controles de governança que este capítulo reutiliza.

- GitHub Copilot Documentation, agent mode, arquivos de instrução do repositório e o seletor de modelos que este capítulo governa.
- Azure AI Foundry Documentation, governança de produção para agentes corporativos: identidade, RBAC e observabilidade.

- What is Azure AI Content Safety?, a camada de filtragem de conteúdo reutilizada como guardrail de agente, não reconstruída.
- Model Context Protocol, o padrão aberto para catalogar e dar escopo às ferramentas que um agente pode chamar.

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 PROVA DE EXECUÇÃO

Locação de veículos em produção, e além.

Dois problemas operacionais na locação de veículos mostram como um design multi-agente fica quando sai do slide e roda em produção. Um é um prazo de multa medido em dias. O outro é um documento fiscal que bloqueia um mês de notas quando um código está errado. Os dois se resolvem nos mesmos quatro movimentos, e os mesmos quatro movimentos valem para outras quatro verticais. Este capítulo é a prova de execução.

Indicação de condutor em multas: quatro agentes contra um prazo de 15 dias

O primeiro caso rodou em locação de veículos, e começou com um relógio legal. Quando um carro alugado leva uma multa de trânsito, a lei dá à locadora 15 dias para indicar o condutor que estava ao volante. Perca essa janela e a empresa absorve a multa mais uma penalidade em cima. O processo manual era frágil. O setor de correspondência recebia o aviso em papel, um operador digitalizava e redigitava os campos em uma planilha, depois enviava e-mail ao cliente que alugou o carro e esperava resposta. Prazos escorregavam com frequência suficiente para virar um item de custo. Quatro agentes especializados substituíram esse fluxo, e eles rodam no Azure AI Foundry.

Cada agente é dono de uma tarefa. Um agente de ingestão usa o Document Intelligence para extrair placa, data da infração, código, valor e prazo do aviso digitalizado. Um agente de consulta cruza SAP e Salesforce para encontrar o locatário ativo. Um agente de notificação envia link do portal, SMS e e-mail. Um agente de acompanhamento observa o relógio: escalona no dia 7, envia um alerta final no dia 12 e encaminha o caso para um paralegal no dia 14. O template de notificação fica atrás de um checkpoint Human-in-the-Loop, então a equipe de operações aprova o texto antes de qualquer mensagem chegar ao cliente. Nada aqui

é exótico. O valor vem de dividir uma cadeia manual frágil em quatro agentes que fazem, cada um, uma coisa estreita e testável.

Um detalhe de implementação importa para custo e repetibilidade. Os quatro agentes foram especificados e construídos com o GitHub Copilot em agent mode. Cada ferramenta que eles chamam é declarada como um servidor MCP, as convenções da locação vivem no .github/copilot-instructions.md, e cada agente escolhe seu modelo no Copilot picker: Haiku 4.5 para extração barata de campos, Sonnet 5 para a lógica de consulta e notificação, e Opus 4.8 onde o timing de escalonamento precisa raciocinar sobre casos de borda. O custo de build é medido em GitHub AI Credits, onde um crédito é um centavo. O mesmo picker e o mesmo arquivo de instruções valem para o fluxo fiscal, e por isso o segundo caso subiu em dias, não em semanas.

Conformidade fiscal de NF-e: três agentes sob controle duplo

O segundo caso tem uma falha mais afiada. Cada encerramento de contrato, devolução de veículo e transferência entre filiais precisa gerar uma Nota Fiscal Eletrônica válida, e as regras vêm da legislação do RICMS, que muda por estado e por categoria de produto na tabela NCM. Um código CFOP errado pode bloquear notas de um mês inteiro. A emissão manual aguenta em volume baixo e quebra em escala. Três agentes conduzem o fluxo, disparados por um evento de ERP do SAP, Oracle ou TOTVS.

O primeiro agente é um classificador fiscal. Ele usa o Azure AI Search sobre o texto do RICMS e a tabela NCM para escolher o CFOP, o CST e a alíquota de ICMS corretos, então a classificação fica ancorada na legislação real em vez de inferida. O segundo agente emite o documento contra a API SEFAZ NF-e 4.0 e trata retentativas, modo de contingência e o caminho de cancelamento. O terceiro agente roda uma reconciliação diária sobre SPED, EFD-Contribuições, os livros de entrada e saída e o relatório de ICMS-ST. O fechamento fiscal mensal não roda sozinho. Ele fica sob um checkpoint de controle duplo: uma pessoa inicia, uma segunda aprova, e o CFO assina com evidência auditada. Alto impacto e baixa reversibilidade merecem o guardrail mais rígido do framework.

O padrão reutilizável: ingerir, classificar, agir, escalonar

Olhe os dois casos e o mesmo esqueleto aparece. Um agente ingere um documento ou um evento. Um agente classifica contra regras que o negócio já codifica, seja um registro de locatário ou um código fiscal. Um agente age, enviando um aviso ou emitindo um documento fiscal. Um agente escalona quando um prazo se aproxima ou uma aprovação humana é exigida. Ingerir, classificar, agir, escalonar. O padrão é deliberadamente sem graça, porque sem graça é o que sobrevive a uma auditoria e a uma revisão de incidente em produção.

O que muda entre os casos não é a forma, é o guardrail. O fluxo de multas revisa um template e escalona em um calendário fixo. O fluxo de NF-e roda uma reconciliação diária e trava o fechamento mensal atrás de controle duplo. As duas decisões saem direto do framework Human-in-the-Loop: mapeie cada ação por impacto e reversibilidade, depois coloque o checkpoint onde uma ação autônoma errada custaria mais do que a revisão. O padrão te dá os agentes. O framework te diz onde um humano ainda precisa estar.

> DEFINIÇÃO
>
> Ingerir, classificar, agir, escalonar. Quatro agentes estreitos, um checkpoint Human-in-the-Loop posicionado por impacto e reversibilidade. Comece com um caso de uso e um time, prove o valor em produção, depois reutilize o esqueleto na próxima vertical.

Quatro verticais sobre o mesmo padrão

O padrão não fica só na locação de veículos. Em finanças, um agente de documentos lê arquivos de onboarding, um agente de pontuação de risco cruza nomes com listas de sanções, e um agente de escalação encaminha ao compliance officer, com controle duplo na aprovação de KYC e AML. Em saúde, um agente de extração clínica se junta a uma camada RAG de regras de planos, e um médico revisa antes de qualquer envio de autorização prévia. O tempo de autorização cai de

dias para horas, e cada decisão é registrada para auditoria. Os mesmos quatro movimentos, regras diferentes, um checkpoint diferente.

Varejo e manufatura repetem de novo. No varejo, um agente de triagem classifica o motivo da devolução, um agente de reembolso roda a verificação de política, e um agente de logística agenda a coleta, com o checkpoint Human-in-the-Loop reservado para exceções acima do limite de valor do SKU. Na manufatura, um agente de telemetria observa sinais de IoT, um agente de diagnóstico propõe ordens de serviço, e um agente de agendamento aloca técnicos, com o gerente de planta aprovando qualquer ordem maior que quatro horas. Quatro verticais, um esqueleto: ingerir, classificar, agir, escalonar, com o checkpoint onde o custo de errar é mais alto. Essa é a prova. O padrão saiu de uma demo para produção na locação, e transfere sem reescrita.

Referências

Fontes primárias para as plataformas, ferramentas e controles de governança por trás dos casos em produção.

- Azure AI Foundry documentation, o runtime em que rodam os quatro agentes de locação e os três agentes de NF-e.
- GitHub Copilot documentation, o agent mode, o .github/copilot-instructions.md e o model picker usados para construir os agentes.
- Model Context Protocol, o padrão pelo qual cada ferramenta de agente é declarada e escopada.
- Azure AI Content Safety overview, os controles de conteúdo e segurança que sustentam os checkpoints Human-in-the-Loop.
- Microsoft Copilot Studio documentation, o caminho low-code para a superfície conversacional dos agentes de processo de negócio.

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 ROTEIRO DE ADOÇÃO

Do primeiro piloto à frota em produção: cinco fases, critérios de saída explícitos.

Um piloto que funciona não é um programa em produção. A distância entre um agente ajudando um time e uma frota governada servindo a empresa inteira se mede em fases, não em funcionalidades. Este capítulo apresenta o roteiro de cinco fases que percorre doze meses, os critérios de saída que impedem um piloto de ir ao ar como demo, e a disciplina de custo que mantém uma frota solvente quando ela roda em GitHub AI Credits.

As cinco fases: Descoberta, Piloto, Endurecimento, Escala, Operação

O caminho do primeiro piloto até uma frota de agentes de nível produção percorre cinco fases ao longo de doze meses. Cada fase muda o que você constrói e o que você precisa provar antes de avançar. A Descoberta acontece no mês um. O Piloto, nos meses dois e três. O Endurecimento, dos meses quatro a seis. A Escala, dos meses sete a nove. A Operação começa no mês dez e não termina. A sequência é deliberada, e você não pode pular uma fase sem transferir o custo dela para uma fase posterior, onde ele é sempre maior e mais difícil de pagar.

A Descoberta mapeia casos de uso candidatos por valor e esforço, obtém a aprovação dos stakeholders e seleciona um candidato ao piloto. O Piloto implanta um único agente com um único time e executa uma avaliação em loop fechado antes que qualquer outra pessoa tenha acesso. O Endurecimento implementa RBAC e trilhas de auditoria, define metas de SLA e tetos de custo, e conecta checkpoints de Human-in-the-Loop em cada ação que toca dados regulados ou estado irreversível. Essas três primeiras fases são como uma ideia se torna algo que um comitê de revisão aceita aprovar para usuários reais.

A Escala constrói o registro de ferramentas, introduz orquestração multi-agente e abre onboarding self-service para que novos times tragam seus próprios casos de uso sem um engenheiro de plataforma no loop de cada um. A Operação estabelece as disciplinas que mantêm uma frota saudável por anos: FinOps, monitoramento de drift, simulações de troca de modelo e revisões trimestrais contra as metas de valor que foram escritas lá na Descoberta. A Escala amplia a superfície do programa. A Operação é o que mantém essa superfície honesta.

Critérios de saída explícitos por fase

Cada fase tem um portão de saída, e o portão é um conjunto de condições, não uma data no calendário. A Descoberta encerra quando um caso de uso é escolhido e sua meta de valor está escrita em um número pelo qual alguém será cobrado. O Piloto encerra quando a avaliação em loop fechado passa em seu golden test set e o time patrocinador realmente quer continuar usando o agente. O Endurecimento encerra quando toda ação de alto impacto e baixa reversibilidade fica atrás de um checkpoint de controle duplo e cada chamada que o agente faz é auditada.

As duas falhas que mais vejo são pular o portão depois do Piloto e pular o portão depois da Escala. Pilotar sem endurecer produz uma demo: funciona na sala e desaba na primeira vez que encontra permissões reais, requisitos reais de auditoria e uma fatura real. Escalar sem operar produz dívida: mais agentes, mais times e mais gasto, sem ninguém observando drift ou lendo a conta do mês. Os critérios de saída existem para tornar essas duas falhas visíveis cedo, antes que se acumulem pela frota.

> DEFINIÇÃO
>
> Cada fase tem critérios de saída explícitos. Pilotar sem endurecer produz uma demo. Escalar sem operar produz dívida.

Operar para o longo prazo

A Operação é onde a maioria dos programas falha em silêncio, porque o trabalho é sem glamour e nunca acaba. Quatro disciplinas o sustentam. O FinOps acompanha o gasto por agente, por time e por loop, e mantém cada um contra o teto de custo definido no Endurecimento. O monitoramento de drift observa o declínio lento que atinge agentes do mesmo modo que atinge modelos: um prompt que funcionava seis meses atrás começa a errar, e o golden test set percebe isso antes dos seus usuários. As simulações de troca de modelo ensaiam trocar o modelo por baixo de um agente sem mudar como o agente se comporta.

As simulações de troca de modelo importam porque o modelo por trás de um agente nunca fica fixo por muito tempo. No GitHub Copilot, o seletor de modelos expõe Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro e a família GPT-5.x. Uma nova entrada no seletor é uma chance de cortar custo ou elevar qualidade, mas só se você conseguir trocá-la e reexecutar o golden test set para provar que o agente continua se comportando. Ensaie essa troca de forma programada, não durante um incidente. As revisões trimestrais então fecham o ciclo: meça cada agente contra sua meta de valor original e aposente os que já não justificam seu lugar.

Custo na prática: gasto por loop

Aqui o custo é uma métrica de primeira classe, não uma surpresa de fim de ano. No GitHub Copilot, o uso de agentes é cobrado em GitHub AI Credits, e um crédito vale um centavo. Desde junho de 2026, essa é a unidade para a qual todo relatório de FinOps converte. Acompanhar o gasto por loop significa atribuir a cada ciclo de plan-act-observe um custo em créditos, para que um time veja quais agentes são baratos, quais são caros e quais são caros por um bom motivo.

A disciplina se acumula ao longo de todo o roteiro. O Endurecimento define o teto de custo. A Escala multiplica o número de loops rodando por dia. A Operação lê o medidor e o reporta. Um agente que custa um punhado de créditos por loop é tranquilo em volume de piloto e vira uma linha de custo real em volume de frota, então o orçamento de latência e de tokens que você define por loop no Endurecimento é o mesmo orçamento que mantém a frota solvente na Operação.

Custo por loop, medido em créditos, é o número único que conecta o design de um agente à conta do programa inteiro.

Referências

Fontes primárias para as plataformas, o modelo de cobrança e os controles de governança dos quais este roteiro depende.

- GitHub Copilot Documentation, o agent mode e o modelo de cobrança em GitHub AI Credits que mede o gasto por loop.
- Azure AI Foundry Documentation, o catálogo de modelos e a observabilidade de produção por trás das simulações de troca de modelo e do monitoramento de drift.
- Microsoft Copilot Studio Documentation, o caminho low-code para o onboarding self-service aberto na fase de Escala.
- Model Context Protocol, o padrão aberto por trás do registro de ferramentas catalogado e com escopo construído na Escala.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, a trajetória de adoção enterprise que enquadra o roteiro de doze meses.

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