THE FULL AGENTIC ENGINEERING PLATFORM · PLAYBOOK

Seis disciplinas decidem se os agentes entregam ou vão para o cemitério.

A maioria dos pilotos agênticos morre em produção, não por falta de ambição ou orçamento, mas por falta de engenharia. Este playbook apresenta a pilha completa em seis disciplinas, Cloud, Platform, Context, Intent, Harness e Token Economy, sob a lente do GitHub Copilot. Desde 1 de junho de 2026, quando o GitHub AI Credits tornou o custo por trabalho executado, essa disciplina paga entre 30 e 70 por cento menos na conta.

AUTORA

Paula Silva

CARGO

AI-Native Software Engineer

CONTATO

in/paulanunes

TEMPO DE LEITURA

~ 45 minutos, fim a fim

6

DISCIPLINAS DE ENGENHARIA

6

CAPÍTULOS

8

PADRÕES DE OTIMIZAÇÃO

30-70%

REDUÇÃO DE CUSTO DE TOKENS

CAPÍTULOS

Chapter 00

O cemitério de agentes e as seis disciplinas

Por que pilotos agênticos morrem em produção, as três dívidas que se acumulam, a pilha de seis disciplinas em sequência e a mudança do GitHub AI Credits de junho de 2026 que faz a disciplina se pagar.

Fundações →

Chapter 01

Fundação cloud e engenharia de plataforma

Kubernetes para agentes de vida longa, MCP como camada de protocolo, golden paths e identidade SPIFFE, e a arquitetura CPE de quatro planos com segurança de cadeia de suprimentos.

Substrato →

Chapter 02

Engenharia de contexto

A curva de utilidade côncava e o context rot, níveis de memória com copilot-instructions.md e MCP, seis técnicas que se compõem e o prompt caching.

Cognição →

Chapter 03

Intent engineering

A dívida de intenção e o caso Klarna, os artefatos do Spec Kit, o Spec-Driven Development com portão humano e hooks de custo zero, e a lacuna planner-coder.

Estratégia →

Chapter 04

Harness engineering

Agente é modelo mais harness, a taxonomia de guias e sensores, quatro padrões de produção, o playbook de sete passos e dois caminhos sobre uma fundação Microsoft.

Modelo operacional →

Chapter 05

Economia de tokens, FinOps e adoção

Roteamento de modelos no seletor do Copilot precificado em AI Credits, FinOps para IA, oito padrões de otimização, Azure AI Foundry e o roteiro de 30, 60, 90 dias.

Financiamento →

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 O PROBLEMA DO CEMITÉRIO

O cemitério de agentes e as seis disciplinas.

A maioria dos pilotos agênticos nunca chega à produção. A lacuna não é de ambição nem de orçamento. É de engenharia. Um agente que funciona no demo morre em produção porque o time pulou camadas: cloud, plataforma, contexto, intenção, modelo operacional e economia de tokens. Este capítulo nomeia por que os pilotos morrem, apresenta as seis disciplinas em sequência e enuncia os dois fatos que fazem a disciplina se pagar.

Por que pilotos morrem em produção: três dívidas que se acumulam

Pilotos agênticos não morrem porque o modelo é fraco. Morrem porque o time entregou sem uma fundação de engenharia coerente, e três dívidas se acumulam por baixo do demo. A primeira é a dívida técnica, a conhecida: patches, dependências frágeis, testes quebradiços. Um estudo em larga escala de Liu e colegas analisou 304.362 commits gerados por IA em 6.275 repositórios e encontrou proporções maiores de dívida de requisitos e de testes do que em código escrito por humanos. A assistência de IA não remove a dívida. Ela muda a forma da dívida.

A segunda dívida é a cognitiva, e é mais perigosa porque a revisão não a pega. Os desenvolvedores aceitam código gerado sem entender por que funciona. Storey chama isso de rendição cognitiva. A base de código vira conhecimento órfão: ela roda, mas ninguém no time consegue modificá-la com segurança porque ninguém entende o raciocínio por trás. O ganho de produtividade no primeiro dia vira um passivo de manutenção no nonagésimo dia.

Dívida de intenção: a falha que parece sucesso

A terceira dívida é a de intenção, a mais nova e a mais insidiosa. O agente foi feito para a métrica X, sob a restrição Y, com a premissa oculta Z. Quando as condições

de produção mudam, o agente continua otimizando para a coisa errada. Não está quebrado. Só nunca foi especificado por completo. A dívida de intenção é a razão de um agente passar em todos os testes e ainda assim fazer o trabalho errado, e é por isso que a especificação, não o código, é o artefato que precisa ser revisado primeiro.

A pilha de seis disciplinas, em sequência

Uma pilha que sobrevive em produção cobre seis disciplinas, e a ordem importa. Cloud vem primeiro: Kubernetes e o Model Context Protocol dão aos agentes um substrato que mantém estado ao longo de sessões longas, em vez de tratar cada chamada como um request web stateless. Platform vem em segundo: golden paths, uma plataforma interna de desenvolvimento e templates do Backstage embutem controles de segurança e FinOps em cada agente por padrão, para que a governança não seja um exercício de papel. Context vem em terceiro: o menor conjunto de tokens de alto sinal de que o agente precisa, montado por memória, MCP e recuperação, e pré-computado no CI em vez de reconstruído a cada tarefa.

Intent vem em quarto: o Spec Kit torna o objetivo legível por uma constituição, uma especificação e um plano de implementação, para que o agente otimize para o resultado que você realmente quer. O modelo operacional vem em quinto, a disciplina Harness: cadência, propriedade, evidência e revisão que transformam o framework de slides em software entregue, por Three Horizons para times alinhados à Microsoft ou por Open Horizons para o stack CNCF. A economia de tokens vem em sexto: FinOps para agentes, com orçamentos por sessão, agente e repositório, e Azure AI Foundry para roteamento de modelos. Pule qualquer camada e o piloto entra para o cemitério.

O fato de junho de 2026: a cobrança passou de intenção para trabalho executado

Em 1 de junho de 2026, o GitHub Copilot parou de cobrar por request. Os Premium Request Units se aposentaram. O GitHub AI Credits assumiu o lugar, e a conversão é simples: 1 AI Credit equivale a US$0,01. Tokens de input, output e cached passam a ser contabilizados na taxa de API publicada de cada modelo no Copilot picker, do

Haiku 4.5 para trabalho trivial ao Sonnet 5 para código e ao Opus 4.8 para planejamento, ao lado do Claude Fable 5, do Gemini 3.1 Pro e dos modelos GPT-5.x. Code completions e Next Edit Suggestions seguem incluídos no plano e não consomem créditos.

A mudança é estrutural, não cosmética. Sob os Premium Request Units, o custo era por intenção: um request, uma cobrança, não importa quanto trabalho ele disparava. Sob o GitHub AI Credits, o custo é por trabalho executado. Uma sessão em agent mode que faz fan-out por um repositório custa o que de fato consome. Isso torna a disciplina de tokens um controle financeiro, não um luxo. Toda técnica deste playbook, do contexto pré-computado no CI, ao roteamento de modelos no Copilot picker, ao prompt caching, aparece agora direto na conta.

Os números que justificam o esforço

O caso da plataforma se apoia em evidência, não em um deck de fornecedor. A Deloitte pesquisou 3.235 respondentes em 24 países e encontrou que 75% das empresas planejam IA agêntica em dois anos, mas apenas 34% relatam usar IA para transformar profundamente o negócio. A lacuna entre planejar e transformar é dívida de engenharia, e é previsível. Um estudo causal de Carnegie Mellon usando o dataset AIDev encontrou que agentes de codificação sem governança adicionam 39% de complexidade cognitiva e 18% de avisos de análise estática, e que essa dívida se acumula de forma persistente em vez de se autocorrigir.

A disciplina se paga. Times que tratam tokens como recurso finito, aplicam context engineering, roteiam para o menor modelo que passa no eval e instrumentam custo pagam entre 30% e 70% menos do que times que rodam o GitHub Copilot como caixa-preta. Em cargas agênticas especificamente, os padrões de otimização aplicados se acumulam para uma redução na mesma faixa. A diferença entre o cemitério e a plataforma não é talento. É disciplina aplicada, em sequência, ao longo de seis camadas.

DEFINIÇÃO

O playbook inteiro em uma frase: pilotos morrem de três dívidas que se acumulam, seis disciplinas de engenharia em sequência mantêm agentes vivos em produção, e desde 1 de junho de 2026, quando o GitHub AI Credits tornou o custo por trabalho executado, essa disciplina paga entre 30% e 70% menos na conta.

Referências

Fontes primárias para o padrão de falha, a pilha de seis disciplinas e os dois fatos que justificam a plataforma.

- Deloitte, State of AI in the Enterprise 2026, a pesquisa por trás da lacuna: 75% das empresas planejam IA agêntica, apenas 34% relatam transformação profunda.
- Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild, a evidência de dívida técnica em 304.362 commits gerados por IA em 6.275 repositórios.
- AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development, a medição causal de como agentes de codificação mudam a saída de engenharia.
- GitHub Docs, Usage-based billing for GitHub Copilot: GitHub AI Credits, a mudança de cobrança de junho de 2026: 1 AI Credit equivale a US$0,01, custo por trabalho executado.
- CNCF, Platforms Whitepaper, a referência para a disciplina de plataforma e os golden paths.
- Spec Kit, toolkit for Spec-Driven Development, a ferramenta por trás da disciplina de intenção: constituição, especificação e plano de implementação.
- Anthropic, Effective context engineering for AI agents, orientação fundamental para a disciplina de contexto.

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 SUBSTRATO E GOVERNANÇA

Fundação cloud e engenharia de plataforma.

As duas primeiras camadas da pilha decidem se um agente chega ou não à produção. A infraestrutura cloud dá às cargas agênticas um lugar para rodar. A engenharia de plataforma decide o que essas cargas têm permissão de fazer. Este capítulo cobre as duas: como o Kubernetes muda quando a carga é um agente de vida longa em vez de um request stateless, por que o Model Context Protocol reduz o custo de integração, como golden paths e identidade de workload transformam governança de documento em código aplicado, e como a arquitetura CPE de quatro planos e os controles de cadeia de suprimentos sustentam a pilha agêntica.

Kubernetes para cargas agênticas

O Kubernetes se tornou o runtime padrão para IA em produção. Duas de cada três organizações que rodam IA em escala o usam como plataforma primária de orquestração. Ele não venceu por ter sido feito para machine learning. Venceu porque já resolvia cargas distribuídas e stateful em escala enterprise. A Cloud Native Computing Foundation então estendeu essa base para IA. Em março de 2026, o Kubernetes AI Conformance Program certificou mais de 70% de plataformas adicionais, um sinal de que o Kubernetes padrão precisa de extensões deliberadas para rodar agentes com segurança.

O que muda é o perfil da carga. Um job de treinamento roda até terminar. Um serviço de inferência responde a um request e retorna. Um agente não faz nem um nem outro. Sessões agênticas duram horas, mantêm KV cache entre passos e fazem fan-out para subagentes. Três capacidades de infraestrutura atacam isso diretamente. O redimensionamento de pod in-place deixa um agente passar de planejamento leve para uso intensivo de tools sem reiniciar, então o pod continua agendado e só a reserva muda. O escalonamento consciente de carga entende a coordenação entre agentes e quebra as esperas circulares que prendem o escalonamento FIFO padrão.

O sandboxing dá a cada agente isolamento em nível de sistema operacional ou de VM, uma fronteira que se mantém mesmo quando o agente roda código que não escreveu. Ao redor da computação ficam os requisitos operacionais: containers de menor privilégio, segredos com escopo e a pilha de observabilidade MELT de métricas, eventos, logs e traces, para que as decisões que um agente toma continuem auditáveis em vez de opacas.

MCP como a camada de protocolo

Sem um protocolo comum, cada agente integra cada ferramenta nos próprios termos. Isso são N vezes M integrações ponto a ponto, e a contagem cresce com agentes e ferramentas ao mesmo tempo. O Model Context Protocol reduz isso para N mais M. Cada agente fala MCP uma vez, cada ferramenta expõe um servidor MCP uma vez, e qualquer agente alcança qualquer servidor. O MCP carrega três primitivas: tools que produzem efeitos colaterais, resources que expõem dados e prompts que moldam a interação. O GitHub Copilot em agent mode consome servidores MCP do mesmo jeito, o que significa que a integração que você constrói para a plataforma é a integração que o Copilot usa.

O valor arquitetural é a fronteira, não o formato do fio. Um agente não acessa um banco de dados ou uma API externa diretamente. Ele fala MCP com um servidor que media o acesso, e um gateway MCP de propriedade da plataforma centraliza autenticação, escopo e telemetria à frente desses servidores. Política, rate limits e classificação de conteúdo ficam no gateway, aplicados sem mudar o código do agente. O MCP é deliberadamente estreito. Conecta agentes a dados, ferramentas e serviços, e não tenta ser dono de observabilidade, identidade de workload ou controle de acesso. Essa contenção é o que o mantém combinável com o OpenTelemetry para traces e com o SPIFFE para identidade. No início de 2026, o protocolo governava mais de 10.000 servidores ativos, com 97 milhões de downloads mensais de SDK, e passou para a Agentic AI Foundation na Linux Foundation, que ultrapassou 170 organizações membras até março de 2026.

Golden paths e identidade

Engenharia de plataforma virou item obrigatório. A avaliação DORA 2025 relatou que cerca de 90% das empresas já operam uma plataforma interna, e o Backstage, a plataforma de desenvolvimento interna open-source, detém perto de 89% de participação entre adotantes. A razão é governança. Sem golden paths, cada time reconstrói os mesmos controles, e cada agente entrega com autenticação, telemetria e manuseio de segredo próprios. Com golden paths entregues como scaffolds do Backstage, templates de IDP e registros de servidor MCP, controles de segurança e de custo já vêm compilados. Times são compliant por padrão e optam por sair só com uma razão declarada.

Um arquivo .github/copilot-instructions.md é um golden path no mesmo sentido. Ele codifica as convenções que o Copilot deve seguir para que o agente comece dentro das guardas em vez de adivinhar. Os quatro pilares de controle de plataforma, golden paths, guardrails, redes de segurança e portões de revisão manual, valem para agentes exatamente como valem para pessoas. Agentes viram cidadãos de primeira classe da plataforma, com o próprio RBAC, as próprias cotas de recurso e os próprios orçamentos de custo.

Identidade e política como código

Identidade é a outra fundação. Agentes precisam de identidades de workload SPIFFE, não de service accounts compartilhados. Cada agente carrega um SPIFFE ID que propaga do chamador ao servidor MCP à fonte de dados, e tokens são criados por sessão com escopo estreito e TTL curto, para que um agente nunca exceda o que o usuário solicitante poderia fazer. Política como código fecha o ciclo. Regras escritas para OPA ou Cedar vivem em repositórios, versionam junto ao código que governam e passam por revisão de pares como qualquer outro artefato de infraestrutura. O log de auditoria de que um revisor precisa é consultável sem ajuda de engenharia.

A arquitetura de quatro planos e a segurança da cadeia de suprimentos

O Cognitive Platform Engineering dá à pilha uma forma de referência. Organiza a plataforma em quatro planos construídos sobre Kubernetes, Terraform e OPA. Um data plane roda cargas de agente e servidores MCP em pods com GPU. Um control plane cuida da orquestração pela API do Kubernetes, GitOps e Argo. Um cognition plane carrega roteamento, política e artefatos de intenção como especificações do Spec Kit e a CONSTITUTION do projeto. Uma camada de observabilidade coleta spans, logs e dados de custo pelo OpenTelemetry.

Os planos fecham um ciclo sentir-raciocinar-agir. A plataforma sente padrões de carga, raciocina sobre configuração e age pelo control plane, para que um agente que passa de planejamento para geração de código encontre recursos antes de um desenvolvedor abrir um chamado.

Protegendo a cadeia de suprimentos agêntica

A superfície de segurança é mais ampla que a segurança de aplicação. Uma análise de 295 GitHub Security Advisories que referenciam componentes de LLM encontrou vulnerabilidades de cadeia de suprimentos em 44% deles, à frente de agência excessiva com 20% e injeção de prompt com 18%. Os controles são conhecidos e baratos. Combine ao menos dois scanners de GitHub Actions, já que nenhum scanner sozinho cobre toda classe de fraqueza, e a varredura mediana custa 2,71 segundos por workflow, então não há desculpa de desempenho para pular. Gere proveniência SLSA e um SBOM no pipeline, assine artefatos e trate toda mudança gerada por agente como não confiável até a política liberar. A cobrança reforça a disciplina. Desde 1 de junho de 2026, o GitHub Copilot cobra em GitHub AI Credits a um crédito por US$0,01, medido nos tokens de input, output e cached que cada modelo consome, então um agente perdulário aparece como linha de custo e um bem escopado não.

DEFINIÇÃO

As camadas um e dois em uma frase: o Kubernetes dá aos agentes um runtime feito para sessões de vida longa e stateful, e a engenharia de plataforma dá a esse runtime as suas regras. Substrato antes de autonomia, protocolo em vez de ponto a ponto, identidade por sessão, política como código e verificações de cadeia de suprimentos em todo pipeline.

Referências

Fontes primárias para a fundação cloud, a camada de protocolo, os golden paths e a segurança da cadeia de suprimentos agêntica.

- CNCF, Platforms Whitepaper, a definição de referência de plataformas internas e golden paths.
- Cognitive Platform Engineering: 4-Plane Reference Architecture, o modelo de quatro planos sobre Kubernetes, Terraform e OPA.
- Model Context Protocol: Design Patterns and Security Considerations, estrutura do protocolo e o padrão de gateway de propriedade da plataforma.
- LLM-Enabled OSS Vulnerabilities in GitHub Security Advisories, a fatia de cadeia de suprimentos nos advisories da pilha agêntica, 44% de 295.
- Unpacking Security Scanners for GitHub Actions, por que dois scanners superam um, a 2,71 segundos de mediana por workflow.
- GitHub Docs, Usage-based billing for GitHub Copilot: GitHub AI Credits, como o consumo de tokens é medido desde 1 de junho de 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

02 A CAMADA DE COGNIÇÃO

Engenharia de contexto: o que o agente sabe.

Engenharia de contexto é a disciplina de montar o menor conjunto de tokens de alto sinal que maximiza a qualidade da saída de um agente. Não é encher a janela com tudo que talvez ajude. É o oposto: curadoria feita com bisturi. No GitHub Copilot em agent mode, o contexto que o agente lê decide tanto a resposta quanto a conta. Este capítulo cobre a curva de utilidade, os níveis de memória, seis técnicas que se compõem e o prompt caching.

A curva de utilidade côncava: mais contexto não é melhor

A intuição comum é que mais contexto sempre ajuda o modelo. Mais informação, melhor resposta. Essa intuição está errada. A relação entre a quantidade de contexto e a qualidade da saída é côncava com pico, não crescente de forma monotônica. Adicionar tokens irrelevantes tem efeito neutro no início e negativo depois. Isso é context rot. Três mecanismos o provocam. A diluição de sinal vem primeiro: para uma pergunta sobre um módulo, injetar cinquenta arquivos de outro módulo obriga o modelo a filtrar ruído antes de conseguir focar. A janela de atenção é finita: mesmo modelos com janela de um milhão de tokens aplicam um gradiente de atenção, então tokens no meio recebem menos atenção do que tokens no início e no fim, e a diluição empurra o material relevante para a zona de baixa atenção. O custo sobe sem benefício: mais tokens sempre custam mais dinheiro, e se a qualidade não aumenta, o gasto é desperdício puro.

A regra a aplicar antes de adicionar qualquer coisa

A Anthropic enuncia a regra sem rodeios: use o menor conjunto de tokens de alto sinal que maximiza a probabilidade do resultado desejado. Não todo o contexto possível. Não o contexto que pareceu relevante à primeira vista. O mínimo que sustenta o resultado correto. Antes de injetar contexto, faça três perguntas. Esta

informação é necessária para esta tarefa específica? Ela pode ser recuperada sob demanda em vez de injetada de antemão? Estou adicionando porque ajuda, ou porque parece uma garantia? A resposta de garantia costuma ser o sinal de que o context rot está sendo introduzido.

Níveis de memória e MCP: guias que o agente lê primeiro

O contexto vive em dois níveis, e confundi-los é caro. A janela de contexto é a memória de trabalho de uma única requisição. É volátil, custosa e finita, de duzentos mil a um milhão de tokens conforme o modelo, e cada turno relê tudo. A memória persistente é informação armazenada fora do contexto, em vector store, key-value store ou banco de dados, e recuperada sob demanda. Ela acumula indefinidamente sem inflar a janela, o armazenamento custa centavos por gigabyte, e a recuperação só custa quando consultada. A pergunta-teste decide onde um fato pertence: isto está em uso ativo nesta interação, ou é referência potencialmente útil? Uso ativo vai para a janela de contexto. Referência potencialmente útil vai para a memória persistente. A recuperação acontece pelo Model Context Protocol (MCP), a interface padrão que o agente usa para alcançar um store, uma tool ou um resource sem uma integração sob medida para cada um.

Arquivos de instrução são contexto curado por humanos

A forma concreta de contexto curado no fluxo do desenvolvedor é o arquivo de instrução. O AGENTS.md na raiz do repositório e o .github/copilot-instructions.md são lidos automaticamente no início de uma sessão, antes de qualquer tarefa. Um estudo sobre AGENTS.md mediu o efeito: um arquivo curado por humanos reduz o runtime em 28,64% e o uso de tokens em 16,58%. Um arquivo gerado por LLM tem desempenho 3% pior do que nenhum arquivo. A curadoria humana é obrigatória. O arquivo .github/copilot-instructions.md é a forma do GitHub Copilot, lida pelo Copilot Chat e pelo Copilot coding agent e renderizada na interface do VS Code. Mantenha esses arquivos enxutos, de cinquenta a duzentas linhas, e atualize-os em pull requests quando as convenções mudarem. Eles codificam onde as coisas ficam, as convenções que a base de código segue e o que o agente nunca deve tocar.

Seis técnicas que se compõem e cortam o custo recorrente de tokens

Seis técnicas se compõem sobre a curadoria, cada uma medida. Compaction é o modelo comprimindo a própria história de sessão em um resumo estruturado que mantém decisões, arquivos alterados e erros, e descarta tentativas fracassadas e enchimento. Medida em um benchmark de código, entregou 22,7% de redução média de tokens, até 57% em sessões muito longas, com 0% de perda de acurácia. O tool-result clearing ataca o maior consumidor de tokens em sessões longas: a leitura de um único arquivo pode injetar trinta mil tokens, e resultados de tool antigos permanecem no contexto e são pagos a cada chamada seguinte se não forem limpos. Mantenha os últimos cinco a dez resultados e substitua o resto por um marcador curto.

Os subagentes E-mem isolam contexto por fan-out: um lead despacha subagentes que exploram cada um uma fatia e retornam apenas um resumo, o que alcançou a mesma qualidade de tarefa com 70% menos tokens. A memória persistente move fatos estáveis para fora da janela; um estudo destilou 4.182 conversas em uma camada compacta com 11x de compressão mantendo a recuperação intacta. O SkillReducer enxuga arquivos de skill de forma programática, 39% de redução no corpo e 48% de redução na descrição. A sexta técnica, prompt caching, é a maior alavanca isolada e ganha sua própria seção. Essas técnicas se empilham: a compaction encolhe a história, o clearing remove resultados obsoletos, os subagentes isolam a exploração, a memória persistente descarrega a referência, e o SkillReducer afina as definições que o agente sempre lê.

> DEFINIÇÃO
>
> Seis técnicas que se compõem: compaction (22,7% de redução média de tokens, 0% de perda de acurácia), tool-result clearing (mantenha os últimos cinco a dez resultados), subagentes E-mem (70% menos tokens), memória persistente (11x de compressão na recuperação), SkillReducer (39% de corpo menor, 48% de descrição menor) e prompt caching (até 90% de economia de input). Cure o menor conjunto de alto sinal e depois reutilize-o.

Prompt caching: reutilizar a KV cache de um prefixo estável

Toda chamada de modelo roda o mecanismo de atenção sobre todos os tokens, o trabalho mais caro da requisição, e armazena o resultado intermediário, a KV cache de atenção, na memória da GPU. Numa requisição posterior com o mesmo prefixo de tokens, o provedor recupera essa KV cache em vez de recomputá-la. O modelo pula o trabalho de processar aqueles tokens de novo. Em cargas conversacionais, isso reduz o custo de input em até 90% sem mudar uma única linha de prompt. A cache opera por hash de prefixo, byte a byte, então qualquer mudança em qualquer byte invalida a cache daquele ponto em diante.

A disciplina é uma regra: conteúdo estável primeiro, conteúdo dinâmico depois. Coloque o system prompt, as definições de tool e o contexto estático no início e marque-os como cacheáveis; mantenha a mensagem do usuário e tudo que muda no fim. Um system prompt alterado invalida o prefixo inteiro, então versione-o com cuidado. Sob os GitHub AI Credits, tokens cacheados são cobrados à taxa publicada de cada modelo, onde um crédito equivale a um centavo, então um prefixo estável não é só mais rápido, é um item de custo economizado a cada chamada repetida. Ao rotear pelo seletor do Copilot, a mesma disciplina de prefixo vale, quer a chamada vá para o Haiku 4.5 num passo trivial, o Sonnet 5 para código, ou o Opus 4.8 para um plano.

Referências

Fontes primárias para a curva de utilidade, a evidência dos arquivos de instrução, o padrão de recuperação e o modelo de cobrança.

- Anthropic, Effective context engineering for AI agents, a fonte da regra: o menor conjunto de tokens de alto sinal que maximiza a probabilidade do resultado desejado.
- GitHub Docs, Customizing GitHub Copilot with instruction files, como o .github/copilot-instructions.md é lido e aplicado em agent mode.
- Spec Kit, toolkit for Spec-Driven Development, o padrão de artefatos de especificação curados por humanos que ancoram agentes custom.

- GitHub Docs, Usage-based billing and GitHub AI Credits, como tokens cacheados e não cacheados são medidos, onde um AI Credit equivale a um centavo.

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 A CAMADA ESTRATÉGICA

Intent engineering: o que o agente deve fazer.

Context engineering torna um agente preciso. Não faz o agente mirar no alvo certo. É essa a lacuna que a intent engineering fecha. Um agente do GitHub Copilot em agent mode pode carregar o repositório inteiro no contexto e ainda otimizar para um resultado que a organização nunca escolheu. Este capítulo é a camada estratégica da pilha. Ele mostra como codificar o que o agente deve fazer em artefatos versionados: CONSTITUTION.md para valores, SPECIFICATION.md para requisitos testáveis e IMPLEMENTATION_PLAN.md para escopo. Depois mostra por que a interface entre planejamento e codificação é onde a maioria das falhas multi-agente começa.

Dívida de intenção: contexto completo, sem alvo, resultado errado

Volto sempre ao caso Klarna do terceiro trimestre de 2025 porque ele mostra exatamente o que quebra. A Klarna implantou um agente com tudo o que se quer do ponto de vista de infraestrutura: dados detalhados de clientes, especificações de produto, regras de precificação, histórico de execução. O contexto era abrangente. Em semanas, o agente passou a otimizar para métricas que a organização nunca quis priorizar. Quando alguém percebeu o desalinhamento, o sistema já tinha incorrido em custo operacional e reputacional. A lição é direta: context engineering não impede um agente de otimizar para a coisa errada. Você pode dar todo o contexto do mundo a um agente, mas se nunca disser o que ele deve otimizar, ele vai otimizar por algo.

A dívida de intenção se acumula em silêncio. Os objetivos, as restrições e o raciocínio por trás deles vivem numa thread de Slack de oito meses atrás, num e-mail perdido num canal de suporte, na cabeça de um único engenheiro. Cada um é um ponto de falha num sistema feito para operar de forma autônoma em escala. Cada geração de trabalho assistido por IA sem artefatos explícitos de intenção

alarga a distância entre o que o sistema deveria fazer e o que ele faz. O artigo do tema nomeia três dívidas que se acumulam juntas: técnica, cognitiva e de intenção. A dívida de intenção é a mais insidiosa porque o agente não está quebrado. Ele só nunca foi especificado por completo.

> DEFINIÇÃO
>
> Um agente bem contextualizado sem intenção escrita é uma flecha com mira excelente e nenhum alvo. O contexto diz ao agente como o mundo funciona. A intenção diz onde ele deve mirar.

Os artefatos do Spec Kit: valores, requisitos, escopo

A intent engineering se materializa em três artefatos, e o Spec Kit os torna legíveis. O CONSTITUTION.md é um documento human-first de princípios, guardrails e valores organizacionais não negociáveis. É escrito para humanos lerem e estruturado para máquinas aplicarem. Uma constituição declara uma hierarquia do que importa mais quando surgem trade-offs, por exemplo, que otimização de custo nunca tem precedência sobre privacidade de dados. Não é um checklist de cada decisão. Num repositório com GitHub Copilot, esses mesmos princípios chegam ao agent mode pelo arquivo .github/copilot-instructions.md, de modo que os valores viajam junto com o código.

O SPECIFICATION.md codifica requisitos num formato testável e legível por máquina. Ele traduz um princípio constitucional em uma condição que o agente pode verificar antes de agir. Se a constituição diz preservar transparência, a especificação diz: quando uma limitação do sistema for descoberta, escale para um revisor humano antes de prosseguir. O IMPLEMENTATION_PLAN.md decompõe o trabalho em tarefas atômicas e marca quais podem rodar em paralelo. O plano vira uma fronteira de controle. O agente sabe exatamente o que pode modificar, e tudo fora do plano está fora de escopo por padrão.

DEFINIÇÃO

Três arquivos, três funções. O CONSTITUTION.md fixa os valores. O SPECIFICATION.md fixa os requisitos testáveis. O IMPLEMENTATION_PLAN.md fixa o escopo. Juntos, são a espinha dorsal do Spec-Driven Development.

Spec-Driven Development: o portão, os hooks, a queda de defeitos

O Spec-Driven Development organiza o trabalho do agente em fases, com um controle rígido entre elas. Primeiro, um humano e um planner redigem o CONSTITUTION.md e o SPECIFICATION.md em Ask mode, sem escrita de arquivos. Depois refinam a especificação em um IMPLEMENTATION_PLAN.md. Então vem o portão de aprovação humana. O agente não entra em agent mode, onde tem permissão de escrita, até um humano aprovar o plano. Isso não é uma fronteira leve. É imposto pela camada de orquestração. O portão move o julgamento humano para cima, para dentro da especificação, onde um desalinhamento ainda é barato de corrigir.

Hooks: enforcement a zero AI Credits

Os hooks aplicam o plano a custo zero de tokens. Um hook preToolUse é um comando de shell que intercepta uma escrita e a bloqueia quando o arquivo-alvo não está no IMPLEMENTATION_PLAN.md. Ele roda na camada de orquestração, não no modelo, então não consome nenhum GitHub AI Credit. O agente propõe, o hook aplica, o humano decide se altera o plano. Pesquisa sobre SDD constitucional constatou que ele reduz defeitos de segurança em 73% frente ao desenvolvimento assistido por IA sem restrições. O mecanismo não é uma especificação perfeita. É que a revisão explícita do plano antes da implementação expõe o desalinhamento enquanto ainda dá para corrigir. Roteie a fase de planejamento para um modelo mais forte no Copilot picker, como Opus 4.8 ou Claude Fable 5, e a implementação com escopo para o Sonnet 5.

DEFINIÇÃO

O portão humano entre especificação e implementação é o controle mais importante do fluxo. Os hooks o tornam barato de manter: um condicional de shell na camada de orquestração, zero tokens de modelo, zero AI Credits.

A lacuna planner-coder: a especificação é a interface

Um estudo de falhas de sistemas multi-agente constatou que 75,3% delas se originam na lacuna entre o planner e o coder. Quando o planner escreve uma especificação ambígua ou incompleta, o coder gera código inseguro ou incorreto. Isso não é só um problema de qualidade. É um vetor de segurança. Um atacante que consiga influenciar o que o planner produz, via prompt injection ou manipulação de contexto, pode fazer o coder gerar código vulnerável sem nunca tocar no coder.

Esse é o argumento empírico mais forte para tratar o SPECIFICATION.md como engenharia de verdade, não como papelada. A especificação é a interface entre planejamento e codificação. Quando essa interface é explícita, legível por máquina e revisada por humanos, você fecha a lacuna onde três quartos das falhas começam. Quando ela é implícita, uma tarefa vaga ou um prompt casual, você deixa essa superfície de falha aberta. A mesma base de evidência mostra que o modelo é um motor de resultado maior que o framework, então invista na especificação e na escolha de modelo no Copilot picker antes de investir na escolha de framework. A intent engineering é a camada estratégica. A harness engineering, no próximo capítulo, é o modelo operacional que a executa.

DEFINIÇÃO

75,3% das falhas de sistemas multi-agente começam na interface planner-coder. A especificação é essa interface. Torne-a explícita, legível por máquina e revisada por humanos, e você fecha três quartos da superfície de falha.

Referências

Fontes primárias para a dívida de intenção, os artefatos do Spec Kit, o Spec-Driven Development e a evidência empírica sobre custo, complexidade e a interface de especificação.

- GitHub, Spec Kit: a toolkit for Spec-Driven Development, o toolkit que torna CONSTITUTION.md, SPECIFICATION.md e IMPLEMENTATION_PLAN.md artefatos concretos e versionados.
- GitHub Docs, Usage-based billing for individuals: GitHub AI Credits, como o trabalho do agente é medido em GitHub AI Credits, um crédito equivalente a um centavo de dólar desde 1 de junho de 2026, e por que hooks na camada de orquestração consomem zero créditos.
- arXiv, Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild, evidência em larga escala de que o código gerado por IA desloca a dívida para requisitos e testes, a matéria-prima da dívida de intenção.
- arXiv, AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development, o impacto medido dos coding agents, incluindo a complexidade cognitiva que agentes sem governança adicionam às organizações de engenharia.
- Deloitte, State of AI in the Enterprise 2026, a lacuna de transformação que a dívida de intenção alarga: três em cada quatro empresas planejam IA agêntica, cerca de um terço relata transformação profunda.
- Anthropic, Effective context engineering for AI agents, a disciplina sobre a qual a intent engineering se apoia, o contexto torna o agente preciso, a intenção diz onde ele deve mirar.

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 EXECUÇÃO E OPERAÇÕES

Harness Engineering: o modelo operacional.

Um agente é um modelo mais um harness. O modelo raciocina. O harness é tudo ao redor: tools, memória, estado, gates, sensores e guias. Troque o modelo no picker do GitHub Copilot e o agente continua se comportando como o harness permite. Este capítulo transforma o stack em um modelo operacional: a taxonomia de guias e sensores, quatro padrões de produção, sete passos de implementação e dois aceleradores sobre uma fundação Microsoft.

Agente é igual a modelo mais harness

A fórmula que nomeou o campo no início de 2026 é compacta: um agente é um modelo mais um harness. O modelo fornece a capacidade de raciocínio. O harness é todo o resto, as tools, a memória, o estado, o contexto, a validação, a segurança, o ciclo de vida, os gates, os sensores e os guias. No GitHub Copilot o modelo é a parte plugável. Você o escolhe no picker, Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro ou GPT-5.x, e o harness permanece constante. Beren Millidge formulou isso em 2023: um modelo cru é uma CPU sem sistema operacional, e o harness é o SO que a torna útil.

Martin Fowler e Birgitta Böckeler deram ao harness sua taxonomia mais clara em março de 2026. Um harness se divide em guias, que orientam o agente, e sensores, que o verificam. Guias são os arquivos que o agente lê primeiro: .github/copilot-instructions.md, AGENTS.md, system prompts versionados, documentos de restrição arquitetural e descrições ricas de tools. Sensores observam o que o agente fez. Sensores determinísticos são baratos e rápidos: linters customizados, type checkers, pre-commit hooks, testes estruturais e gates de CI que bloqueiam o merge quando qualquer verificação falha. Sensores inferenciais são baseados em LLM: code review como juiz e análise semântica de pull request. Um estudo de arquivos AGENTS.md descobriu que arquivos curados por humanos reduziram o runtime em 28,64% e o uso de tokens em 16,58%, enquanto arquivos gerados por

LLM tiveram desempenho 3% pior do que nenhum arquivo. A disciplina é a curadoria, não o tamanho do arquivo.

> DEFINIÇÃO
>
> Agente é igual a modelo mais harness. O modelo é o componente plugável que você seleciona no picker do GitHub Copilot. O harness é o que seu time constrói e mantém: guias que orientam o agente e sensores que o verificam. Construa um agente e o que você construiu foi um harness apontado para um modelo.

Padrões de produção

Quatro padrões se repetem em produção. O ReAct loop é o mais simples: o agente raciocina, escolhe uma tool, observa o resultado e itera até atingir o objetivo ou esgotar o orçamento. Use-o para tarefas curtas com verificação clara de sucesso: triagem de issues, sumarização, geração de relatório. O padrão initializer mais coding agent trata trabalho que se estende por muitas sessões. Um initializer roda uma vez e cria a estrutura: um init script, um arquivo de progresso como log de sessão a sessão, um primeiro commit e uma feature list de itens rastreáveis. Um coding agent então roda em cada sessão seguinte: lê o progresso, escolhe a próxima feature, implementa, testa com Playwright, atualiza o log e faz o commit. O estado vive em arquivos versionados, então a memória de longo prazo fica legível e auditável.

O padrão Planner-Generator-Evaluator, da Anthropic em março de 2026, divide o trabalho em três papéis. O Planner expande um prompt curto em uma especificação e deixa deliberadamente os detalhes de implementação em aberto. O Generator implementa em sprints e antes assina um sprint contract com o Evaluator: uma definição de pronto compartilhada. O Evaluator dirige a aplicação pelo Playwright como um usuário real e falha o sprint se algo quebra. A pesquisa encontrou que 75,3% das falhas de multi-agente se originam na interface entre planner e coder, e o Evaluator a torna explícita. O quarto padrão é a garbage collection contínua. Agentes de fundo varrem o repositório em cadência atrás de imports que cruzam camadas, duplicação e desvio, depois abrem pequenos pull requests de refatoração que

se revisam em menos de um minuto e fazem auto-merge. A OpenAI trata dívida técnica como um empréstimo de juros altos, paga continuamente em vez de em surtos dolorosos.

O playbook de sete passos

Os sete passos são sequenciais e cumulativos. Primeiro, escolha o caso de uso certo. Um bom caso tem critério de sucesso verificável, trabalho repetitivo que justifica o investimento, impacto material e risco contornável. Critérios vagos, trabalho pontual e risco ilimitado são onde os pilotos morrem. Segundo, escreva o arquivo de instrução como um mapa, não uma enciclopédia. Para o GitHub Copilot isso é o .github/copilot-instructions.md. Mantenha-o perto de 100 linhas e deixe-o apontar para fontes mais profundas em um diretório docs estruturado. Terceiro, instrumente trajetórias desde o dia um: o registro completo de uma sessão, prompts, chamadas de tool, saídas, erros e tempos. Sem ela você não depura, e cada sessão capturada vira dado de avaliação. Quarto, plante sensores determinísticos antes de qualquer juiz LLM. Um linter customizado para regras arquiteturais, um type checker em cada pull request, pre-commit hooks e um gate de CI obrigatório pegam a maioria das falhas de forma barata.

Quinto, imponha a arquitetura mecanicamente. Codifique a direção de dependência como um linter em vez de um parágrafo. No stack Microsoft, combine Bicep com Azure Policy, analisadores Roslyn e branch protection no GitHub. Este é o passo que separa pilotos de produção. Sexto, adicione loops de autoverificação: sprint contracts, orquestração test-first, automação de navegador Playwright e loop detection middleware que aborta em ações repetidas. A LangChain saltou do rank 30 para o rank 5 no Terminal Bench 2.0 adicionando sensores e autoverificação com o modelo mantido constante. A peça faltante costuma ser um sensor, não um modelo melhor. Sétimo, execute o loop de melhoria contínua. Observe uma falha em uma trajetória, classifique a causa raiz, faça o agente escrever a correção sob revisão, adicione um teste de regressão e audite trajetórias futuras. Cada falha se torna estruturalmente impossível de repetir. Três engenheiros usando essa disciplina entregaram cerca de um milhão de linhas de código de produção sem código escrito por humanos.

Dois caminhos sobre uma fundação Microsoft

A fundação sob cada engajamento é constante. Azure é a cloud, GitHub Enterprise carrega o controle de fonte e o DevOps, Entra Agent ID dá a cada agente uma identidade gerenciada com escopos e trilha de auditoria, e Azure AI Foundry com Microsoft Agent Framework é a plataforma de IA. O GitHub Copilot entrega as primitivas de harness diretamente. O arquivo .github/copilot-instructions.md é o guia do repositório; arquivos .instructions.md com escopo por caminho e arquivos .prompt.md reutilizáveis o estendem. Custom chat modes guardam as personas do time, agentic workflows carregam loops e gates, e a configuração de servidor MCP expõe as tools. O GitHub Copilot coding agent trata o trabalho multi-sessão, e a automação de code review atua como sensor inferencial nos pull requests. A cobrança roda em GitHub AI Credits desde 1 de junho de 2026: um crédito equivale a um centavo de dólar, e tokens de input, output e cached são medidos na taxa publicada de cada modelo, então o custo acompanha o trabalho executado, não os assentos.

Dois aceleradores materializam o mesmo harness. Three Horizons usa Red Hat Developer Hub sobre Azure Red Hat OpenShift para clientes com investimento prévio em Red Hat ou requisitos fortes de suporte. Open Horizons usa Backstage sobre Azure Kubernetes Service para clientes Azure-native. Ambos implementam a mesma disciplina e chegam ao mesmo destino Horizon 3; só mudam a plataforma interna de desenvolvimento e o cluster. O mapeamento de modo de falha para camada de harness torna a revisão de incidentes mecânica: imports que cruzam camadas apontam para um linter de direção de dependência, conclusão prematura para um sprint contract e um evaluator separado, um agent loop para loop detection middleware, um segredo vazado para um pre-commit hook e um sandbox sem acesso a segredos, e um comando destrutivo para uma política de aprovação obrigatória. Nenhum deles se resolve com um prompt melhor. Cada um precisa de uma peça estrutural do harness, e as primitivas de agent mode no GitHub Copilot fornecem a maioria delas de fábrica.

Referências

Fontes primárias para a fórmula do harness, a taxonomia de guias e sensores, os quatro padrões de produção e o modelo operacional de sete passos.

- Mitchell Hashimoto, My AI Adoption Journey, origem da disciplina: engenheirar o harness para que o agente nunca repita um erro.
- Ryan Lopopolo, OpenAI, Harness engineering in an agent-first world, o playbook de sete passos, a garbage collection contínua e cerca de um milhão de linhas de código sem código escrito por humanos.
- LangChain, Improving Deep Agents with harness engineering, do rank 30 ao rank 5 no Terminal Bench 2.0 mudando o harness com o modelo mantido constante.
- Birgitta Böckeler, Harness engineering for coding agent users, martinfowler.com, a taxonomia de guias e sensores sobre a qual este capítulo se apoia.
- Anthropic, Harness design for long-running application development, o padrão Planner, Generator e Evaluator para sessões autônomas de várias horas.
- Microsoft Agent Framework 1.0, a camada gerenciada Agent Harness sobre a fundação compartilhada Azure e GitHub.

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 FINANCIAMENTO E FECHAMENTO

Economia de tokens, FinOps e adoção: financie a plataforma, aposente o cemitério.

A plataforma está construída. A conta continua chegando. Em 1 de junho de 2026, o GitHub Copilot deixou de cobrar por request e passou a cobrar pelos tokens que cada modelo de fato consome, então escolha de modelo, cache e roteamento viram itens de linha que um time de finanças consegue ler. Este capítulo financia a plataforma: como rotear modelos no seletor do Copilot, como operar FinOps para IA, como cortar 25 a 45 por cento sobre o baseline e como adotar em escala enterprise.

Roteamento de modelos no seletor do Copilot: pague cada modelo pela sua taxa de API

O seletor do Copilot expõe um modelo por interação, e o agent mode lê essa escolha em cada tarefa. Roteie por complexidade. O Haiku 4.5 cuida das tarefas triviais: classificação, extração e trabalho em lote de baixa nuance. O Sonnet 5 é o cavalo de batalha para código: debugging, refatorações multiarquivo e revisão profunda. O Opus 4.8 fica reservado para planos: arquitetura e decisões que serão reutilizadas. Cada chamada premium contabiliza tokens de input, output e cached pela taxa de API publicada daquele modelo, convertida a 1 AI Credit igual a US$0,01. Modelos bundled incluídos no plano não gastam créditos.

O roteamento é a alavanca de custo de primeira ordem, e o lugar dele é o controle de versão, não a cabeça de cada dev. Defina o padrão como Sonnet 5 para agent mode e edit mode, fixe o Opus 4.8 no plan mode e desça para o Haiku 4.5 em tarefas em lote bem definidas. Faça commit dessa política em .vscode/settings.json e declare a justificativa em .github/copilot-instructions.md para que o time inteiro rode o mesmo roteamento. O tier premium custa várias vezes o do cavalo de batalha para a mesma sessão, então um padrão de Opus 4.8 é dinheiro gasto sem ganho equivalente em qualidade.

FinOps para IA: Inform, Optimize, Operate

FinOps para IA é a disciplina de instrumentar, medir, otimizar e operar o gasto de tokens com o rigor aplicado a qualquer outro recurso de nuvem. Roda em três fases. Inform captura telemetria por request, modelo, tokens, usuário e feature, via OpenTelemetry, exporta sob o schema FOCUS e mostra tudo em dashboards do Power BI ou Grafana. Sem dados, otimização é chute. Comece pelo showback: mostre a cada time o custo que ele gerou sem cobrar, porque o comportamento muda quando o número fica visível.

Optimize identifica o desperdício específico: prompts ineficientes, modelos superdimensionados, falta de cache, times sem treino. Cada correção tem um payback estimável, mensurável e atribuível. Operate fecha o ciclo com chargeback para os times, detecção de anomalia que sinaliza uma sessão rodando três sigma acima da média e revisão trimestral de orçamento. A maturidade se mede em meses, não em dias. Estabeleça o baseline no primeiro mês, capture quick wins até o terceiro e institucionalize o ciclo de melhoria até o sexto, quando o showback evolui para chargeback.

Oito padrões de otimização, 25 a 45 por cento sobre o baseline

Os números sustentam o argumento. Para uma organização de 500 devs a US$19 por assento, um baseline de US$9.500 por mês, pilotos sem disciplina observaram 50 a 100 por cento de overage após a transição do PRU. Com disciplina aplicada, a mesma organização alcança 25 a 45 por cento de redução sobre o baseline em 60 a 90 dias, acumulando 40 a 70 por cento em cargas agênticas. Para organizações acima de 5.000 devs, a economia anual passa de US$1 milhão. O payback é inferior a 60 dias.

Oito padrões entregam essa redução: roteamento dinâmico por complexidade, cache hierárquico em três níveis, subagent fan-out com hierarquia E-mem, plan-then-execute com um modelo de planejamento mais barato, cache semântico por similaridade de embedding, composição de tools em vez de crescimento de catálogo, MCP enxuto com disclosure progressivo de schemas e SDD de agente custom guiado por Spec-Kit. Cinco anti-padrões anulam os ganhos e precisam sair:

o maior modelo por padrão, um agente onde um workflow basta, o repositório inteiro colado no contexto, pular o prompt caching e pular o harness.

Azure AI Foundry e migração: PTU versus PAYG

O Azure AI Foundry complementa o GitHub Copilot, não o substitui. O Copilot resolve codificação na IDE. O Foundry roda cargas de produção com governança, um catálogo de mais de 1.800 modelos, content safety, RBAC e auditoria dentro do tenant Azure. Ele oferece dois modelos de cobrança. O PAYG cobra por token pelas taxas publicadas sem compromisso, o que serve para carga variável ou com picos. O PTU reserva capacidade de inferência com throughput garantido e latência consistente, o que serve para volume estável e alto. A escolha é por workload, não por organização.

O roteiro de adoção de 30, 60, 90 dias

A heurística é simples: PAYG abaixo de US$5.000 por mês por modelo, PTU acima de US$30.000 e um benchmark no meio. O roteiro segue o dinheiro. Nos primeiros 30 dias, analise o bill preview, configure o cap de paid usage com alertas, roteie modelos no seletor e ligue a telemetria. Dos 30 aos 60 dias, aplique roteamento e cache, faça a curadoria do copilot-instructions.md e conduza o workshop de tech leads. Dos 60 aos 90 dias, institucionalize o showback, conecte o export FOCUS aos dashboards e mova cargas estáveis para o Foundry onde compliance ou economia de PTU justificarem.

A conclusão: disciplina, não talento

Setenta e cinco por cento das empresas planejam IA agêntica. Apenas 34 por cento relatam transformação profunda. A lacuna não é de orçamento e não é de talento. É de disciplina aplicada nas seis disciplinas: Cloud, Platform, Context, Intent, Harness e Token Economy. A economia de tokens é a disciplina que mantém as outras cinco financiadas, porque uma plataforma que ninguém consegue custear é uma plataforma que ninguém opera.

Sob os PRUs, o custo era por intenção, e uma sessão agêntica de quatro horas custava o mesmo que uma pergunta rápida. Sob tokens, o custo é por trabalho executado, e cada otimização aparece na fatura. Times que tratam tokens como recurso finito, com orçamentos por sessão, agente e repositório, pagam 30 a 70 por cento menos pelo mesmo resultado. Essa margem é a diferença entre o cemitério e a plataforma, e ela se compra com prática, não com headcount.

> DEFINIÇÃO
>
> Economia de tokens em uma frase: precifique cada chamada de modelo pela taxa de API publicada em AI Credits, roteie o menor modelo que passa no eval e financie a plataforma com a economia. Disciplina aplicada, não talento, separa o cemitério da plataforma.

Referências

Fontes primárias para a mudança de cobrança, a disciplina de FinOps, as opções do Azure AI Foundry e a evidência de adoção.

- GitHub Docs, Usage-based billing and GitHub AI Credits, a mudança de cobrança: 1 AI Credit igual a US$0,01, com tokens cobrados pela taxa de API publicada de cada modelo.
- GitHub Copilot, Available AI models, o seletor de modelos que toda decisão de roteamento tem como alvo.
- FinOps Foundation, FinOps Framework, as fases Inform, Optimize e Operate aplicadas ao gasto com IA.
- FOCUS, FinOps Open Cost and Usage Specification, o schema padrão para export unificado de custo entre provedores.
- Microsoft Learn, Azure AI Foundry, a plataforma governada para cargas de produção e o catálogo de modelos.
- Microsoft Learn, Provisioned throughput (PTU), a opção de capacidade reservada comparada ao pay-as-you-go.
- Deloitte, State of AI in the Enterprise, a lacuna de adoção: a maioria das empresas planeja IA agêntica, bem menos relata transformação profunda.

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