GOVERNANÇA DE APIS E TOOLING ENTERPRISE · PLAYBOOK

Um plano de controle governa cada API, extensão e agente que chega à workstation.

Ambientes corporativos de desenvolvimento acumulam risco de segurança, compliance e produtividade mais rápido do que qualquer time consegue governar. Este playbook constrói um plano de controle sobre três hubs, Azure API Center para APIs, um VS Code Private Marketplace para extensões e governança Copilot para servidores MCP e agentes, para que nada chegue a uma workstation de desenvolvedor sem vetting. Os artefatos mudam. O contrato não.

AUTORA

Paula Silva

CARGO

AI-Native Software Engineer

CONTATO

in/paulanunes

TEMPO DE LEITURA

~ 45 minutos, fim a fim

3

HUBS DE GOVERNANÇA

6

CAPÍTULOS

6

PERSONAS DE PLATAFORMA

30–50%

SHADOW APIS ENCONTRADAS

CAPÍTULOS

Chapter 00

Dispersão sem plano de controle

Por que ambientes corporativos acumulam risco de segurança, compliance e produtividade ao mesmo tempo, como shadow APIs e extensões sem vetting chegam à workstation, a tese do plano de controle único e as seis personas de plataforma para quem este guia foi escrito.

Conceito →

Chapter 01

Três hubs, uma workstation

O plano de controle integrado e a camada de política compartilhada, o papel de cada hub, os conceitos que separam API Center de API Management, MCP e AllowedExtensions, e como os hubs convergem no VS Code com o GitHub Copilot.

Arquitetura →

Chapter 02

Azure API Center como fonte da verdade

A linha entre API Center e API Management, registro por portal, CLI e CI/CD, metadata obrigatória para compliance, linting Spectral no portão, descoberta de shadow APIs com Dev Proxy e o fluxo no VS Code.

Inventário de APIs →

Chapter 03

O Private Marketplace e a política de extensões

Um registro stateless para extensões internas, públicas rehosted e air-gapped, AllowedExtensions aplicada por Group Policy e Intune, o bootstrap install e a queda de footprint de cerca de 150 para aproximadamente 12 vetadas.

Marketplace →

Chapter 04

Governando MCP, agentes e artefatos do Copilot

Aprovação e distribuição de servidores MCP, agentes customizados e arquivos de prompt, escolha de modelo no seletor do GitHub Copilot, confirmações do agent mode e a arquitetura de segurança que cobre cadeia de suprimentos, prompt injection e auditoria.

AI Tooling →

Chapter 05

Do Dia 0 ao Dia 2, e a governança

Os três cenários de deploy, a sequência do Dia 0 ao Dia 2 que sobe e mantém a plataforma, a matriz de ownership com caminhos de aprovação e cadência de revisão, e o princípio da herança de construir uma vez.

Operações →

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 A DISPERSÃO

Dispersão sem plano de controle: três riscos, uma tese, seis personas.

Ambientes corporativos de desenvolvimento acumulam risco mais rápido do que qualquer time consegue governar. APIs se multiplicam sem inventário central. Extensões, servidores MCP e agentes Copilot chegam às workstations de desenvolvimento sem revisão. Os três riscos que decorrem disso, segurança, compliance e produtividade, não são independentes, e a maioria das organizações não tem uma superfície única a partir da qual endereçar os três ao mesmo tempo. Este capítulo nomeia a dispersão, enuncia a tese que a corrige e identifica as seis personas de plataforma para quem este guia foi escrito.

Três riscos convergentes que não compartilham superfície de controle

Ambientes corporativos acumulam risco em três categorias que se compõem, e a maioria das organizações não tem uma superfície única a partir da qual endereçar as três ao mesmo tempo. Segurança é o primeiro ponto de pressão. Extensões e servidores MCP instalados ad hoc, fora de qualquer processo de aprovação, podem exfiltrar código ou credenciais. O IDE vira o alvo fácil. Está dentro do perímetro, tem acesso profundo ao sistema de arquivos e à rede, e raramente é coberto pelo mesmo modelo de ameaça que a infraestrutura de produção. Compliance é o segundo. Dado sensível chega às APIs de modelo de IA sem autorização, e a auditoria não consegue provar o que saiu do ambiente, quando saiu ou para quem foi enviado. À medida que o desenvolvimento assistido por IA se torna padrão, e que essas chamadas de modelo passam a ser cobradas via GitHub AI Credits, a lacuna de compliance se alarga mais rápido do que a governança manual consegue fechar.

Produtividade é o terceiro risco, e muitas vezes o menos intuitivo. Quando o desenvolvedor não descobre o que já existe, ele reconstrói. Usa endpoints depreciados. Queima horas configurando ferramentas que já existem em algum

outro lugar da organização. O custo da duplicação cresce silenciosamente. Os três riscos não são independentes. Tooling sem vetting impulsiona o uso de shadow APIs. O uso de shadow APIs produz lacunas de fluxo de dados. Lacunas de fluxo de dados geram exposição de compliance. Um plano de controle que endereça apenas uma categoria deixa as outras duas abertas, e é por isso que a correção precisa ser estrutural, não um conjunto de soluções pontuais coladas sobre o ambiente depois do fato.

Shadow APIs, extensões sem vetting, servidores MCP e agentes

A dispersão tem uma forma específica. APIs proliferam sem inventário central, e shadow APIs nascem porque os times não conseguem enxergar o que já existe. Endpoints não documentados rodam em produção, chamados por serviços que nenhum registro conhece. Na prática, uma primeira passada de descoberta contra um gateway ativo costuma revelar de trinta a cinquenta por cento das APIs de produção que não aparecem em registro algum, e a maioria dos times subestima essa fatia pela metade até rodar a descoberta. Essa lacuna é a medida mais afiada de quão distante o inventário derivou da realidade, e é a dívida que toda decisão de governança posterior herda.

O mesmo padrão se repete acima da camada de API. Extensões VS Code são instaladas sem vetting, e uma única base de desenvolvedores costuma se espalhar por cerca de 150 extensões distintas antes de qualquer política estar em vigor. Agentes de IA e plugins Copilot puxam dados para serviços externos sem supervisão. Servidores MCP são configurados ad hoc, um desenvolvedor de cada vez, cada um um novo ponto de integração com suas próprias premissas de confiança. Agentes customizados definidos em arquivos `.github/agents/` e instruções em `.github/copilot-instructions.md` moldam como o GitHub Copilot se comporta em agent mode, então um arquivo de instrução não revisado é uma lacuna de governança, não apenas uma conveniência. Cada um desses tipos de artefato é um canal para dentro da workstation, e cada canal é não governado por padrão.

A tese: nada chega a uma workstation de desenvolvedor sem vetting

A correção é uma regra, e cabe em uma frase: nada chega a uma workstation de desenvolvedor sem passar por um catálogo governado. Três hubs a aplicam. O Azure API Center é o inventário autoritativo para cada API da organização, com linting Spectral no momento do registro e descoberta via Dev Proxy para as shadow APIs já em circulação. O VS Code Private Marketplace é o catálogo curado para cada extensão, internas e públicas rehosted por igual. A governança Copilot cobre cada servidor MCP, agente customizado e arquivo de prompt. Três hubs, um plano de controle.

Os três hubs convergem para uma única workstation de desenvolvimento, VS Code com GitHub Copilot, protegida por uma camada de política compartilhada que aplica AllowedExtensions, RBAC, identidade e log de auditoria. O GitHub Copilot em agent mode lê instruções de `.github/copilot-instructions.md` e chama ferramentas via MCP, então a mesma camada de política que governa extensões e APIs também governa como o agente se comporta. O catálogo não é um portão que atrasa os desenvolvedores. É o caminho rápido, porque o artefato com vetting é o que já está instalado e já é confiável desde o primeiro dia.

Um plano de controle, não três ferramentas

A ideia não é três ferramentas separadas coladas uma na outra. Cada categoria de artefato, APIs, extensões e tooling de IA, recebe seu próprio contrato de governança, mas todo contrato é aplicado pelo mesmo plano de controle. Construa uma vez, e a próxima categoria de artefato, o que vier depois de servidores MCP e agentes Copilot, herda automaticamente a mesma superfície de aplicação. Quando os planos estão conectados, toda superfície voltada para o desenvolvedor gera um evento logado, e essa trilha de auditoria é a diferença entre um piloto em indústria regulada que escala e um que é revogado na próxima auditoria externa.

DEFINIÇÃO

Nada chega a uma workstation de desenvolvedor sem passar por um catálogo governado. O Azure API Center governa cada API, o VS Code Private Marketplace governa cada extensão, e a governança Copilot cobre cada servidor MCP, agente e arquivo de prompt. Os artefatos mudam. O contrato não.

Para quem é este guia: seis personas de plataforma

Este guia foi escrito para os times de plataforma que são donos do plano de controle, não para os desenvolvedores que o consomem. Seis personas se repetem entre implantações corporativas. O platform engineer constrói e opera os três hubs. O IT administrator aplica a política de extensões via Group Policy no Windows e Intune em macOS e dispositivos gerenciados. O security engineer é dono do modelo de ameaça, dos controles de supply chain e da postura de auditoria. O developer enablement lead faz do caminho governado o caminho fácil, de modo que o catálogo curado também seja o mais rápido. O enterprise architect mantém os três hubs coerentes com a plataforma mais ampla. O API program manager trata o inventário como um produto, com donos, classificações e um ciclo de vida.

Cada persona lê o guia de forma diferente, e os capítulos que seguem são estruturados para isso. Platform engineers geralmente começam pela arquitetura e pela sequência operacional. Security engineers vão direto ao modelo de ameaça e ao cenário de indústria regulada. Enablement leads se importam mais com o caminho de onboarding do desenvolvedor. O guia constrói o plano de controle um hub de cada vez, depois conecta os hubs, para que qualquer persona possa entrar pela camada que é sua e ainda ver como ela se liga ao resto da plataforma.

Referências

Fontes primárias para os padrões de governança de inventário, marketplace e tooling de IA deste capítulo.

- Azure API Center, inventário e governança centralizados, o registro autoritativo para cada API, versão, ambiente e deployment.
- Analisar APIs no seu API center, linting e análise com Spectral, como regras de padrão de design bloqueiam APIs não conformes no momento do registro.
- Encontrar shadow APIs com Dev Proxy e API Center, trazendo à superfície APIs não documentadas que os desenvolvedores já chamam antes que virem dívida em produção.
- Vincular um Azure API Center ao Azure API Management, como a verdade do gateway e a verdade do inventário convergem em um único registro.
- Gestão corporativa de extensões no Visual Studio Code, política AllowedExtensions via Group Policy e Intune.
- Usar servidores MCP no Visual Studio Code, o modelo de instalação e distribuição para servidores MCP governados.

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 A ARQUITETURA

Três hubs, uma workstation: o plano de controle integrado.

Três hubs convergem para uma única workstation de desenvolvimento. O Azure API Center governa as APIs, um marketplace privado governa as extensões do VS Code e a governança Copilot governa os artefatos de IA. Uma camada de política compartilhada abrange os três, e nada chega à workstation sem passar por ela. Este capítulo mapeia a arquitetura: o plano de controle, o papel de cada hub, os conceitos que os separam e como eles se encontram dentro do VS Code com o GitHub Copilot.

O plano de controle integrado e a camada de política compartilhada

A arquitetura parte de uma única workstation: o VS Code (Desktop) executando o GitHub Copilot. Todo o resto existe para governar o que chega a essa superfície. O desenvolvimento corporativo acumula risco em três frentes ao mesmo tempo: extensões e servidores MCP sem vetting que podem exfiltrar código, dado sensível fluindo para APIs de modelo sem autorização, e trabalho duplicado quando o dev não encontra o que já existe. Um plano de controle que fecha apenas um desses ciclos deixa os outros dois abertos. A arquitetura integrada fecha os três governando três categorias de artefato por meio de uma única superfície de aplicação.

A camada de política compartilhada é essa superfície. Ela aplica AllowedExtensions, RBAC, identidade e log de auditoria em todos os hubs. AllowedExtensions é uma política enterprise do VS Code que controla quais extensões podem instalar e executar. O RBAC define quem pode ler e registrar artefatos. A propagação de identidade associa um usuário real a cada ação, de modo que os logs de auditoria permanecem atribuíveis. A regra é simples e rígida: nada chega à workstation do desenvolvedor sem passar pelo vetting. A política não muda quando o tipo de

artefato muda. APIs, extensões e ferramentas de IA recebem cada uma seu próprio contrato de governança, e o mesmo plano de controle aplica todos eles.

> DEFINIÇÃO
>
> A regra âncora: nada chega à workstation do desenvolvedor sem passar pelo vetting. Cada categoria de artefato recebe seu próprio contrato de governança, mas todo contrato é aplicado pelo mesmo plano de controle. Os artefatos mudam. O contrato não.

Papéis dos hubs: Azure API Center, Private Marketplace, governança Copilot

Cada hub é dono de uma categoria de artefato. O Hub 01 é o Azure API Center, o inventário autoritativo para cada API da organização. Cada versão, cada definição, cada ambiente e deployment é registrado ali. Pipelines de CLI e CI/CD alimentam o registro automaticamente, então o catálogo reflete o que os times realmente entregam. Campos de metadata customizada capturam ownership e classificação, o que torna o catálogo ao mesmo tempo uma superfície de descoberta e um registro de governança. O linting Spectral executa no momento do registro e bloqueia APIs que violam padrões de design da organização antes do deploy. A descoberta via Dev Proxy traz à superfície APIs não documentadas que os devs já estão chamando, de modo que shadow APIs entram no catálogo antes de endurecerem como dívida de produção.

Marketplace e governança Copilot

O Hub 02 é o VS Code Private Marketplace, um catálogo curado de extensões sob controle da própria empresa. O design é intencionalmente stateless: um container com object storage de apoio, implantável em qualquer região, servindo extensões internas e extensões públicas rehosted. Deploys air-gap mantêm os devs produtivos sem nenhum egress para o marketplace público. A aplicação de políticas usa Group Policy no Windows e Intune em macOS e dispositivos gerenciados. Um bootstrap install garante que todo desenvolvedor abra um VS Code curado no primeiro dia, não uma instalação bruta e não gerenciada.

O Hub 03 é a governança Copilot, que aplica o mesmo princípio aos artefatos de IA. Servidores MCP são registrados em `.vscode/mcp.json` e distribuídos pelo marketplace, cada um com um modelo de confiança documentado. Extensões Copilot são revisadas quanto ao fluxo de dados e adicionadas à allowlist no nível da organização GitHub Enterprise. Agentes customizados vivem em arquivos `.github/agents/`, versionados junto ao código que governam e revisados no mesmo pipeline de pull request. Prompts e skills são curados em um repositório compartilhado com workflow de aprovação antes de chegarem à frota de desenvolvimento.

Conceitos-chave: API Center vs API Management, MCP, AllowedExtensions

Três conceitos carregam a maior parte do peso, e o primeiro é a diferença entre o Azure API Center e o Azure API Management. São serviços separados com trabalhos separados. O API Management é o gateway de runtime: ele fica à frente do tráfego vivo, aplica políticas e roteia requests. O API Center é o inventário e o registro de governança: ele cataloga cada API independentemente de onde ela roda, incluindo APIs que nunca passam pelo gateway. Os dois se conectam, não se sobrepõem. O API Management alimenta o API Center upstream automaticamente, então a verdade do gateway e a verdade do inventário convergem em um único registro. Um time pode registrar uma API no API Center antes de qualquer gateway existir, e depois ligá-la ao API Management quando entrar em produção.

MCP e AllowedExtensions

O segundo conceito é o Model Context Protocol. O MCP é um padrão aberto que permite a um agente chamar ferramentas e fontes de dados externas por uma interface uniforme. Nesta arquitetura, os servidores MCP são a unidade de capacidade de IA: um servidor expõe um conjunto de ferramentas, o GitHub Copilot em agent mode o consulta sob demanda, e o modelo de confiança de cada servidor é documentado antes do deploy. Registrar os servidores em `.vscode/mcp.json` e distribuí-los pelo marketplace transforma uma integração ad hoc em um artefato governado.

O terceiro conceito é o AllowedExtensions, a política enterprise do VS Code que define quais extensões uma instalação gerenciada pode carregar. É o mecanismo por trás do marketplace privado: a política aponta o editor para o catálogo interno e bloqueia qualquer coisa fora dele. Aplicado por Group Policy ou Intune, o AllowedExtensions é o que torna um marketplace curado obrigatório, não apenas recomendado.

Como os hubs convergem no VS Code com o GitHub Copilot

Os três hubs resolvem para o mesmo lugar: o editor em que o desenvolvedor trabalha. O API Center é acessível de dentro do VS Code, então o dev descobre e registra APIs sem sair do IDE. O marketplace privado é de onde as extensões são instaladas. A governança Copilot decide quais agentes, servidores MCP e instruções estão ativos na sessão. A workstation é onde a governança se torna visível para quem faz o trabalho, e onde ela ajuda ou atrapalha.

O GitHub Copilot é a superfície que amarra tudo. No início da sessão ele lê o `.github/copilot-instructions.md`, então as convenções do repositório viajam junto com o código. Em agent mode, ele consulta os servidores MCP que a governança Copilot aprovou. O seletor de modelos deixa o dev escolher o modelo para a tarefa, Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro ou um modelo GPT-5.x, enquanto a organização governa quais modelos e agentes estão disponíveis. O uso é cobrado em GitHub AI Credits, onde um crédito equivale a um centavo, então a atividade dos agentes carrega um custo que a plataforma consegue medir. O retorno operacional aparece no Dia 2: o linting bloqueia APIs não conformes no portão, o Dev Proxy traz shadow APIs à superfície antes da produção, e os logs de auditoria dão aos times de compliance visibilidade contínua. Construa a superfície de aplicação uma vez, e toda nova categoria de artefato a herda.

Referências

Fontes primárias para a arquitetura dos três hubs, a camada de política e os conceitos-chave.

- Azure API Center overview, o hub de inventário autoritativo para cada API.
- Azure API Center key concepts, a distinção entre o API Center e o API Management.
- Link an Azure API Center to Azure API Management, como o gateway alimenta o inventário upstream.
- Enterprise extension management in Visual Studio Code, AllowedExtensions e instalações gerenciadas.
- Use MCP servers in Visual Studio Code, registro de servidores MCP para o GitHub Copilot em agent mode.
- Model Context Protocol Specification, o padrão aberto por trás do hub de capacidade de IA.
- GitHub Copilot custom instructions, como o .github/copilot-instructions.md chega à sessão.
- Using GitHub Copilot with an AI model of your choice, o seletor de modelos na workstation.

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 INVENTÁRIO DE APIS

Azure API Center como fonte da verdade: um registro para cada API.

Uma plataforma de governança é tão honesta quanto seu inventário. Se um time não consegue dizer quais APIs existem, qual versão é canônica e quais tocam dados sensíveis, todo controle acima dessa lacuna vira chute. Este capítulo coloca o Azure API Center na base da stack: como ele difere do API Management, como as APIs são registradas e classificadas, como o linting e a descoberta de shadow APIs mantêm o registro honesto, e como os desenvolvedores chegam a tudo isso de dentro do VS Code.

Traçando a linha entre API Center e API Management

O Azure API Center e o Azure API Management resolvem problemas diferentes, e a plataforma quebra quando um time os trata como um único produto. O Azure API Center é o registro único de cada API da organização. Ele grava quais APIs existem, qual versão é canônica, se uma definição está em conformidade com o guia de estilo e quem é dono de cada entrada. Guarda metadata, versões, definições OpenAPI e ambientes de deployment ao longo de todo o ciclo de vida, do design inicial à depreciação. O Azure API Management é a camada de runtime. É o gateway que publica, protege, aplica rate limit e monitora o tráfego ao vivo do subconjunto de APIs que ele gerencia.

A relação vai em uma direção e depois volta. O API Center conhece todas as APIs. O API Management governa o tráfego das que ele expõe. Os dois se conectam por sincronização: vincule uma instância do Azure API Management e todo o seu catálogo flui para o API Center automaticamente, de modo que a verdade do gateway e a verdade do inventário convergem em um único registro, sem reentrada manual. Uma forma prática de explicar isso a stakeholders é dizer que o API Management é o guarda de trânsito no cruzamento, e o API Center é o setor de

planejamento que sabe que cada rua existe. A linha também importa para o custo. Ao vincular uma instância de API Management nos tiers Standard, Standard v2, Premium ou Premium v2, o plano Standard do API Center vem sem custo adicional enquanto ao menos uma instância qualificada permanecer vinculada.

Registro por portal, CLI e CI/CD, e metadata para compliance

Um registro só ganha confiança quando cada API chega até ele da mesma forma toda vez. O API Center suporta três caminhos de registro que espelham como os times realmente trabalham. O portal do Azure faz o registro manual: adicione a API, defina o tipo (REST, GraphQL, gRPC, SOAP, Webhook), escolha um estágio de ciclo de vida e faça upload da definição OpenAPI. A CLI do Azure roteiriza os mesmos passos com az apic api create, az apic api version create e az apic api definition import-specification, o que torna o registro repetível e revisável. O CI/CD fecha a lacuna. Um workflow do GitHub Actions disparado por mudanças em arquivos openapi registra uma nova versão a cada merge na main, e o catálogo nunca fica atrás do código.

Metadata transforma uma lista em inventário governado

O registro grava que uma API existe. A metadata customizada grava o que compliance e segurança precisam saber sobre ela. Defina as propriedades uma vez com az apic metadata create e exija-as em cada entrada: um dono do time, um booleano para indicar se a API processa PII, um enum de domínio de negócio. Marque as propriedades como obrigatórias e o catálogo para de aceitar APIs órfãs, sem dono e sem classificação de dados. É isso que separa uma superfície de descoberta de um registro de governança. Quando um auditor pergunta quais APIs de produção tocam PII, a resposta é um filtro de metadata, não uma enquete.

Linting Spectral no portão, descoberta de shadow APIs com Dev Proxy

Dois controles mantêm o registro honesto: um bloqueia specs ruins na entrada, o outro encontra as APIs que nunca pediram permissão. O API Center roda linting

embutido movido pelo motor open-source Spectral. Faça upload do guia de estilo da sua organização como um ruleset Spectral, e cada definição OpenAPI é validada contra ele no momento do registro. Definições que quebram regras de nomenclatura, versionamento ou segurança são sinalizadas antes de irem para produção, não pegas na revisão três semanas depois. As regras são as mesmas em todo lugar, então o design de API deixa de depender de quem revisou o pull request.

Shadow APIs são as chamadas que uma aplicação já faz e que não aparecem em nenhum catálogo. São dívida técnica e exposição de compliance ao mesmo tempo. O Microsoft Dev Proxy intercepta o tráfego HTTP da aplicação, grava cada chamada e compara os endpoints observados com o inventário do API Center. O relatório lista cada API chamada mas nunca registrada, com passos de registro sugeridos. Na prática, times que rodam a descoberta pela primeira vez encontram de 30 a 50 por cento das APIs em produção ausentes de qualquer registro. Trazer essas APIs para o API Center antes que se acumulem é a diferença entre um inventário que reflete a realidade e um que documenta intenções.

O fluxo no VS Code, do browse ao grounding do Copilot

O registro só é útil se os desenvolvedores chegarem a ele sem sair do editor. A extensão do Azure API Center para VS Code traz o inventário completo para a Activity Bar. Os desenvolvedores navegam pelas instâncias por assinatura, abrem qualquer definição OpenAPI com syntax highlighting e filtram por metadata com uma query como domain:payments pii:true lifecycle:production. Registram novas APIs clicando com o botão direito em um spec, o que mantém o registro como ato de design, não como limpeza pós-deployment. Geram um cliente tipado em TypeScript, Python, C#, Java ou Go a partir de qualquer definição registrada, para que o código do consumidor case com a versão atual em vez de um palpite escrito à mão. O mesmo ruleset Spectral roda inline enquanto digitam, expondo violações no painel Problems antes de o spec ser enviado.

É aqui que o GitHub Copilot ganha seu lugar. Com a extensão @azure do Copilot habilitada, o agent mode ancora suas respostas no inventário vivo do API Center em vez de adivinhar. Pergunte quais APIs tratam autenticação de clientes, ou quais processam PII, e o Copilot retorna respostas contextuais a partir do registro, não dos

dados de treino. Aponte-o para um controller e ele redige uma spec OpenAPI em conformidade com o estilo registrado, pronta para registrar pela extensão. Exponha o inventário pelo Model Context Protocol e qualquer agente no picker, seja Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro ou GPT-5.x, trabalha a partir do mesmo contexto autoritativo. Registre as regras de grounding em .github/copilot-instructions.md para que o agente de cada desenvolvedor comece da mesma fonte da verdade. O faturamento do GitHub Copilot roda sobre GitHub AI Credits, onde um crédito equivale a um centavo de dólar, então consultas ancoradas que evitam rederivar o catálogo a cada sessão saem mais baratas além de mais precisas.

> DEFINIÇÃO
>
> A regra é simples: uma API é real quando está no API Center, não quando está em produção. O registro é um ato de design, a metadata é obrigatória e não opcional, o linting roda no portão, e o Dev Proxy encontra o que escapou. Tudo o que o desenvolvedor e o agente fazem passa por um único registro.

Referências

Fontes primárias no Microsoft Learn para o API Center como hub de inventário, seus controles de registro e governança, e o fluxo no VS Code.

- Microsoft Learn, What is Azure API Center, o hub central de inventário e governança para cada API da organização.
- Microsoft Learn, Synchronize APIs from Azure API Management, como a verdade do gateway flui para o inventário automaticamente.
- Microsoft Learn, Register APIs with GitHub Actions, registro por CI/CD que mantém o catálogo atualizado a cada merge.
- Microsoft Learn, Enable API analysis and linting, regras Spectral validando cada definição no momento do registro.
- Microsoft Learn, Discover shadow APIs with Dev Proxy, encontrando as APIs que rodam em produção mas não aparecem em nenhum registro.
- Microsoft Learn, Build and register APIs with the VS Code extension, o fluxo de browse, registro e geração de cliente dentro do editor.

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 PARTE II, TOOLING CURADO

O Private Marketplace e a política de extensões: um único canal até o editor.

Um marketplace curado é o segundo plano de controle. O primeiro governa APIs. Este governa o que roda dentro do editor. Cada extensão, cada servidor MCP e cada artefato do GitHub Copilot chega ao desenvolvedor por um único canal que o time de plataforma controla. Este capítulo descreve esse canal: um registro stateless, a policy AllowedExtensions que o aplica, o bootstrap install que o semeia e a redução de footprint que ele produz.

Um registro stateless: internas, públicas rehosted e air-gapped

O Private Marketplace é um catálogo curado de extensões que a empresa controla. Sua arquitetura é stateless por design. O registro roda como um container apoiado em object storage, e pode ser implantado em qualquer região que a organização precise. Ele serve dois tipos de extensão. O primeiro é interno: extensões que a organização constrói para os próprios fluxos, publicadas direto no catálogo. O segundo é público rehosted: extensões do marketplace público do VS Code, baixadas, vetadas e republicadas em storage interno para que o editor nunca precise alcançar a galeria pública.

O design stateless é o que faz o cenário air-gapped funcionar. Em uma rede restrita sem egress público, o marketplace mantém em storage interno cada extensão de que um desenvolvedor precisa, e o editor aponta para o endpoint interno em vez do público. O mesmo registro serve a empresa conectada à nuvem e o banco regulado. Só a fronteira de rede muda. Este é o Hub 02 na arquitetura de três hubs, e fica sob a mesma camada de política que o Azure API Center e a governança Copilot. Nada instala no editor sem passar por ele.

AllowedExtensions aplicada por Group Policy e Intune

Um catálogo é uma sugestão até uma policy torná-lo a única opção. AllowedExtensions é essa policy. É uma configuração enterprise do VS Code que nomeia as extensões que um editor gerenciado tem permissão de instalar, e bloqueia todo o resto. No Windows o time de plataforma a distribui por Group Policy. Em macOS e outros dispositivos gerenciados, ela vai por Intune. O mecanismo muda conforme o sistema operacional. O resultado é idêntico: o editor do desenvolvedor pode instalar o que o catálogo permite e nada além disso.

A policy não é uma tranca que prende as pessoas. Uma extensão fora da allowlist ainda é instalável sob pedido, e o pedido deixa uma aprovação rastreável. Essa distinção importa para a postura de compliance. Em um deploy regulado, cada extensão em cada workstation volta a um registro de aprovação, e um auditor pode ler a lista sem pedir ao time de plataforma que a reconstrua. AllowedExtensions também governa a superfície de IA. Uma extensão do GitHub Copilot, ou qualquer extensão que carregue um servidor MCP, está sujeita ao mesmo portão que um realçador de sintaxe. O catálogo curado é onde vive o tooling de IA aprovado, não um canal separado.

Bootstrap install e o catálogo enterprise recomendado

A policy decide o que é permitido. O bootstrap install decide com o que um desenvolvedor começa. Um bootstrap install é um setup de primeira execução por script que entrega a cada desenvolvedor um VS Code curado no primeiro dia, não um editor bruto que ele configura à mão. Ele aponta o editor para o Private Marketplace, aplica a policy AllowedExtensions e instala o catálogo recomendado. Um recém-chegado abre o editor e encontra os linters, o suporte de linguagem e as extensões internas que o time já combinou.

O catálogo recomendado é onde a lente do GitHub Copilot fica mais nítida. O GitHub Copilot chega nesse install já governado. Seu seletor de modelos vem pré-preenchido com os modelos que a organização aprovou, tirados do mesmo conjunto que todo usuário do Copilot vê: Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5,

Gemini 3.1 Pro e a família GPT-5.x. Seu comportamento é moldado por um .github/copilot-instructions.md versionado, e os servidores MCP que ele alcança são os registrados em .vscode/mcp.json e distribuídos pelo mesmo marketplace. O trabalho em agent mode é cobrado em GitHub AI Credits, onde um crédito é um centavo sob o modelo em vigor desde junho de 2026, e um install governado é o que torna esse gasto atribuível a um time e a um repositório, em vez de uma linha misteriosa na fatura.

Footprint de referência: 150 extensões para cerca de 12 vetadas

O número do footprint é o que se defende em uma revisão. Uma base típica de desenvolvedores acumula por volta de 150 extensões distintas entre todas as workstations, instaladas ad hoc ao longo do tempo sem vetting. Quando AllowedExtensions aplica o catálogo curado, esse número cai para cerca de 12 extensões vetadas mais as internas que a organização publica. Esta é uma redução de referência observada na prática, não uma meta fixada por política. O vetting é o que encolhe a superfície, e a policy é o que a mantém ali.

Um footprint menor é uma superfície de ataque menor. Cada extensão instalada no editor roda com o acesso a sistema de arquivos e rede do desenvolvedor, e é por isso que uma extensão sem vetting é um caminho plausível para exfiltração de código ou credenciais. Cortar de 150 para 12 remove a maior parte dessa exposição sem remover capacidade, porque as 12 são as que os desenvolvedores realmente precisavam. As extensões bloqueadas seguem disponíveis sob pedido, então um desenvolvedor com uma necessidade real abre um pedido e recebe uma aprovação rastreável. O padrão é curado. A exceção é documentada. Esse é todo o princípio de design do marketplace, e é o mesmo contrato que o Azure API Center aplica a APIs e a governança Copilot aplica a agentes.

DEFINIÇÃO

O footprint de extensões em uma linha: de cerca de 150 extensões ad hoc em uma base de desenvolvedores para cerca de 12 vetadas mais as extensões internas, aplicado por AllowedExtensions e reversível por uma aprovação rastreável. Curado por padrão, documentado por exceção, atribuível de ponta a ponta.

Referências

Fontes primárias para a arquitetura do Private Marketplace, a mecânica da política de extensões e a distribuição de tooling de IA descrita neste capítulo.

- Visual Studio Code, Private Marketplace announcement, a referência para o registro stateless que serve extensões internas e públicas rehosted.
- VS Code Private Marketplace, setup and rehosting readme, passos de deploy e rehosting de extensões públicas para redes air-gapped e conectadas.
- Visual Studio Code, Enterprise extension management, a configuração AllowedExtensions que torna o catálogo curado o único conjunto instalável.
- Visual Studio Code, Enterprise policies, como a policy é distribuída por Group Policy no Windows e Intune em dispositivos gerenciados.
- Use MCP servers in Visual Studio Code, como os servidores MCP usam o mesmo canal de distribuição que as extensões curadas.

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 AI TOOLING

Governando MCP, agentes e artefatos do Copilot: o contrato do tooling de IA.

Servidores MCP, agentes customizados, arquivos de prompt e instruções do Copilot são código que roda dentro da workstation do desenvolvedor. Eles leem o sistema de arquivos, chamam APIs internas e agem em nome do desenvolvedor. Trate-os como artefatos de tooling, com o mesmo contrato de aprovação, distribuição e auditoria de qualquer extensão ou qualquer API. Este capítulo define esse contrato para a camada de tooling de IA e fecha com a arquitetura de segurança que a sustenta.

Servidores MCP: aprovação, distribuição e credenciais

O Model Context Protocol conecta um agente do GitHub Copilot a ferramentas e dados externos durante uma conversa. Sem ele, o Copilot lê apenas o código no editor. Com ele, um agente em agent mode pode consultar o Jira, ler um banco de dados, chamar uma API interna ou buscar no Confluence. O MCP usa um modelo cliente-servidor. O VS Code é o cliente. Cada sistema expõe um servidor MCP que declara suas capacidades como tools, resources e prompts. O agente decide quais tools invocar a partir do pedido que recebe. Servidores rodam localmente via stdio ou remotamente via HTTP. Para ferramentas organizacionais, servidores remotos compartilhados são o modelo preferido: uma configuração, um único lugar para gerenciar credenciais.

Um servidor MCP é um artefato de tooling do desenvolvedor. Governe-o com o mesmo rigor de uma extensão ou de uma API. Um servidor que conecta a um banco de dados interno é tão sensível quanto as credenciais do banco que ele guarda. Todo servidor passa por um portão de segurança antes de chegar ao catálogo aprovado: código-fonte ou relatório do fornecedor revisado, permissões no mínimo

necessário, read-only sempre que possível, servidores remotos em HTTPS com autenticação, e nenhuma transmissão de conteúdo de requisição para destinos não autorizados. Só então o servidor é adicionado ao catálogo da organização e liberado para distribuição.

Distribuição e tratamento de credenciais

Distribua os servidores aprovados onde os desenvolvedores já trabalham. Um arquivo `.vscode/mcp.json` versionado no repositório dá a cada clone as mesmas ferramentas, sem configuração manual. Uma configuração base entregue via Intune ou JAMF pré-carrega os servidores aprovados pela organização em cada workstation gerenciada, e os desenvolvedores adicionam servidores específicos do repositório por cima. Credenciais nunca ficam em `settings.json` ou `mcp.json` como strings literais. Referencie-as como variáveis de ambiente e obtenha-as de um gerenciador de segredos como o Azure Key Vault no início da sessão. A configuração declara qual segredo precisa; o segredo em si permanece no vault. Exposição de credenciais e prompt injection são os dois riscos nomeados do modelo de confiança do MCP, e ambos são tratados aqui.

Agentes customizados, arquivos de prompt e instruções

O GitHub Copilot tem vários tipos de artefato, e cada um o estende de forma diferente. Um agente customizado é um arquivo Markdown em `.github/agents/`, selecionado no seletor de agentes do Copilot Chat. Ele codifica conhecimento de domínio do time, convenções e acesso a ferramentas com escopo, direto no repositório. Um arquivo de prompt é um template reutilizável `.prompt.md`, invocado por um slash command para tarefas complexas recorrentes. As instruções do Copilot ficam em `.github/copilot-instructions.md` e se aplicam automaticamente a toda sessão de Chat no repositório. Como os três são arquivos, eles são versionados junto ao código que governam e revisados no mesmo pipeline de pull request.

Coloque um agente no diretório `.github/agents/` do repositório `.github` da organização e ele fica disponível em todos os repositórios, sem configuração por repositório. O mesmo vale para arquivos de prompt e instruções em nível de organização. É justamente esse alcance que exige um contrato de revisão dos

artefatos. Agentes customizados são revisados pelo time de developer enablement antes do merge. Arquivos de prompt que chamam comandos de terminal exigem revisão de segurança. Um inventário central de agentes, prompts e instruções evita a proliferação descontrolada, que é a versão de tooling de IA da dispersão de extensões.

Instruções como superfície de fundamentação

O arquivo de instruções é onde o repositório diz ao agente como o código realmente funciona: os padrões de API, a stack de tecnologia, as convenções de nomenclatura e de erros. É também onde vivem as barreiras de segurança. Instrua o agente a tratar o texto retornado por ferramentas externas como dado não confiável, a mostrar o comando completo antes de modificar infraestrutura e a nunca fazer push para um remoto sem confirmação explícita. Instruções são fundamentação, não decoração. Um agente que as lê produz código que pertence ao repositório, em vez de código que apenas compila.

Escolha de modelo e confirmações do agent mode

O GitHub Copilot expõe um seletor de modelos. O desenvolvedor escolhe o modelo por tarefa a partir do conjunto disponível: Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro e a família GPT-5.x. Um agente customizado pode fixar um modelo padrão em sua definição, e o desenvolvedor pode sobrepô-lo no seletor. Case o modelo com o trabalho. Um refactor grande em muitos arquivos recompensa um modelo mais forte; uma edição curta e bem delimitada roda bem em um mais rápido. O faturamento é medido em GitHub AI Credits, onde um crédito equivale a um centavo, em vigor desde 1 de junho de 2026. A escolha de modelo é, portanto, uma decisão de custo tanto quanto de qualidade, e pertence à mesma conversa de governança que todo o resto desta camada.

O agent mode é onde o modelo para de sugerir e começa a agir: ele edita arquivos, roda comandos de terminal e invoca tools MCP. Essa autonomia precisa de freio. Configure o VS Code para exigir confirmação do desenvolvedor antes de ações sensíveis: comandos de terminal, escritas de arquivo e invocações MCP. Deixe essas confirmações em always para operações de escrita. O prompt de confirmação

é o ponto em que um humano lê o plano e aprova ou rejeita cada passo. É a diferença entre um agente que ajuda e um agente que surpreende.

> DEFINIÇÃO
>
> A regra do agent mode: confirmação é obrigatória para toda escrita. Comandos de terminal, escritas de arquivo e invocações MCP exigem, cada um, aprovação humana explícita. O desenvolvedor vê exatamente o que o agente fará antes que ele faça.

Arquitetura de segurança: modelo de ameaça, cadeia de suprimentos, auditoria

A camada de tooling de IA adiciona superfícies de ataque, e cada uma precisa de uma mitigação nomeada. Uma extensão maliciosa pode exfiltrar código: a policy AllowedExtensions e um marketplace privado com revisão de segurança a bloqueiam. Uma resposta de tool MCP pode carregar um prompt injection: confirmação para ações de escrita e barreiras do agente o contêm. Uma extensão Copilot de terceiros pode receber contexto de código: uma allow list em nível de organização e uma revisão de fluxo de dados a governam. Um agente customizado pode executar uma ação destrutiva: políticas de confirmação e instruções de barreira o restringem. A segurança da cadeia de suprimentos é defesa em profundidade. A policy bloqueia extensões não aprovadas, o marketplace privado rehospeda apenas pacotes revisados, e uma auditoria regular compara o que está instalado com a lista aprovada.

Prompt injection é o risco característico da camada agêntica. Acontece quando o dado retornado por uma tool, o corpo de um ticket do Jira ou uma página do Confluence, contém instruções que manipulam o agente. As mitigações são em camadas: exigir confirmação antes de toda ação do agente, treinar os desenvolvedores para ler o plano completo antes de aprovar, e adicionar instruções de barreira que marcam toda saída de ferramenta externa como não confiável. Acima dos controles fica a trilha de auditoria. Quando os planos estão conectados, cada chamada de API, cada invocação MCP e cada ação de agente gera um evento logado no Microsoft Purview ou no SIEM escolhido. A trilha de auditoria é a diferença

entre um piloto em indústria regulada que escala e um que é revogado na próxima auditoria externa.

> DEFINIÇÃO
>
> Trate o texto retornado por ferramentas externas como dado não confiável. Um ticket do Jira, uma página do Confluence ou um registro de banco de dados pode carregar instruções dirigidas ao agente, e o agente não deve segui-las. Portões de confirmação mais instruções de barreira em copilot-instructions.md são a defesa.

Referências

Fontes primárias para governança de MCP, agentes customizados, escolha de modelo e a postura de segurança do Copilot.

- Model Context Protocol Specification, o padrão que conecta agentes do Copilot a ferramentas e dados externos.
- Use MCP servers in Visual Studio Code, configuração e distribuição via mcp.json.
- GitHub Copilot Custom Agents, o formato de agente Markdown em .github/agents/.
- GitHub Copilot Custom Instructions, a superfície de fundamentação e barreira em copilot-instructions.md.
- Using GitHub Copilot with an AI model of your choice, escolha de modelo no seletor do Copilot.
- GitHub Copilot Trust Center, privacidade de dados e postura de segurança para o Copilot enterprise.

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 EXECUTAR E GOVERNAR

Do Dia 0 ao Dia 2, e o framework de governança.

Uma arquitetura vale o que valem seu rollout e seu estado estável. Este capítulo cobre os três cenários de deploy que o plano de controle suporta, a sequência do Dia 0 ao Dia 2 que sobe e mantém a plataforma, e o framework de governança que define quem é dono do quê e como cada artefato chega a um desenvolvedor. O objetivo de design é uma plataforma na qual uma nova categoria de artefato entra sem um novo projeto de governança.

Três cenários de deploy: integrado completo, air-gapped, regulado

A mesma arquitetura suporta três formatos de deploy, e a escolha depende da postura de rede e da carga regulatória, não de um design diferente. No cenário integrado completo, a empresa está conectada à nuvem. O design de API acontece no VS Code, registra no Azure API Center, sincroniza com o Azure API Management e é consumido de volta no IDE. O VS Code Private Marketplace e o GitHub Copilot rodam online. O agent mode lê o .github/copilot-instructions.md e chama servidores MCP registrados no .vscode/mcp.json. Esse é o baseline, e exercita cada hub de ponta a ponta.

O cenário air-gapped remove o egress de rede pública. O marketplace rehosteia extensões públicas em object storage interno, os servidores MCP rodam em endpoints internos, e o GitHub Copilot Enterprise alcança a organização por private link. Os desenvolvedores seguem produtivos sem nenhum caminho para o marketplace público. O cenário regulado, para finanças e setor público, adiciona profundidade à postura de compliance em vez de mudar a topologia: retenção de log de auditoria, rótulos de classificação nos artefatos, binários assinados e propagação de identidade por request. O auditor é independente do time de plataforma. Toda ação é atribuível e todo artefato é rastreável.

> DEFINIÇÃO
>
> Uma arquitetura, três posturas. Integrado completo exercita cada hub online. Air-gapped remove o egress e rehosteia internamente. Regulado adiciona retenção, classificação, artefatos assinados e identidade por request. A topologia não muda; os controles apertam.

Do Dia 0 ao Dia 2: planejar, implantar, operar

O Dia 0 é planejamento e standup da plataforma. O Azure API Center é provisionado como o inventário autoritativo. O VS Code Private Marketplace é implantado como um container stateless com object storage por trás. As políticas de governança do GitHub Copilot são configuradas. Identidade e RBAC são conectados nos três hubs antes de um único desenvolvedor passar pelo onboarding. O Dia 0 também decide o orçamento de AI Credits para o uso do Copilot: desde 1 de junho de 2026, as requisições premium do GitHub Copilot são cobradas em AI Credits, onde um crédito é um centavo de dólar, então a escolha no model picker entre Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro e a família GPT-5.x é uma decisão de custo que o time de plataforma planeja de antemão.

O Dia 1 é a sequência de deploy e o onboarding da frota. Bootstrap installs distribuem instâncias de VS Code curadas via Group Policy no Windows e Intune em dispositivos macOS gerenciados, para que todo desenvolvedor abra um IDE governado no primeiro dia, e não uma instalação bruta. Os registros de APIs começam a fluir via CI/CD para o API Center. O Azure API Management é vinculado para que a verdade do gateway e a verdade do inventário convirjam em um único registro. Os modelos de confiança dos servidores MCP são documentados, um por servidor, antes de os servidores chegarem à frota. Agentes customizados em .github/agents/ e instruções em .github/copilot-instructions.md são versionados junto ao código que governam.

Dia 2: estado estável e onboarding contínuo

O Dia 2 é operação em estado estável, e é onde a plataforma prova seu valor. O linting Spectral roda no momento do registro e bloqueia APIs não conformes no

portão. O Dev Proxy traz à superfície shadow APIs que os desenvolvedores já estão chamando e as incorpora ao catálogo antes que virem dívida de produção. Os logs de auditoria dão aos times de compliance visibilidade contínua. O onboarding não para no Dia 1: cada nova API, extensão, servidor MCP e agente Copilot entra pelos mesmos portões, então a frota cresce sem enfraquecer a governança.

> DEFINIÇÃO
>
> O Dia 0 sobe o API Center, o Private Marketplace, as políticas do Copilot, a identidade e o RBAC, e define o orçamento de AI Credits. O Dia 1 empurra o VS Code curado via Group Policy e Intune, faz os registros de APIs fluírem por CI/CD e documenta os modelos de confiança MCP. O Dia 2 roda os portões: o linting bloqueia APIs não conformes, o Dev Proxy captura shadow APIs, os logs de auditoria seguem contínuos.

O framework de governança: ownership, caminhos de aprovação, cadência de revisão

A governança começa com uma matriz de ownership. Cada categoria de artefato tem um dono nomeado e um aprovador definido. As APIs são de propriedade do time produtor e aprovadas contra padrões de design organizacionais aplicados pelo Spectral. As extensões são de propriedade do time de plataforma e adicionadas à allowlist pelo marketplace. Os servidores MCP carregam um modelo de confiança documentado e um dono responsável pelos dados que podem alcançar. As extensões Copilot são revisadas quanto ao fluxo de dados e adicionadas à allowlist no nível da organização GitHub Enterprise. Os agentes customizados são de propriedade do time cujo código eles tocam. O RBAC no API Center e os controles no nível de organização no GitHub tornam a matriz aplicável em vez de aspiracional.

Cada categoria de artefato tem seu próprio caminho de aprovação, e o caminho corresponde ao risco do artefato. Uma API é aprovada quando passa no linting no registro e carrega a metadata obrigatória de ownership e classificação. Uma extensão é aprovada quando é revisada e publicada no marketplace privado. Um servidor MCP é aprovado quando seu modelo de confiança está documentado e seu endpoint é vetado. Um agente Copilot ou um arquivo de instruções em

.github/copilot-instructions.md é aprovado no mesmo pipeline de pull request do código que governa, revisado pelas mesmas pessoas, sob as mesmas branch protections. O caminho de aprovação é code review, não uma fila de tickets separada.

Cadência de revisão

A cadência de revisão impede que os controles decaiam. Regras de linting e esquemas de classificação são revisados em um ciclo fixo conforme os padrões evoluem. Os modelos de confiança dos servidores MCP são re-revisados quando o escopo de um servidor muda. A allowlist é auditada para que nada permaneça sem um dono. A política do model picker é revisitada conforme novos modelos entram no picker e conforme chegam os dados de consumo de AI Credits, porque o custo das requisições premium é um sinal de governança, não apenas uma linha de finanças. A cadência é a diferença entre uma governança que se sustenta e uma que se corrói em silêncio.

> DEFINIÇÃO
>
> O ownership nomeia quem é responsável. Os caminhos de aprovação roteiam cada artefato pelo controle que corresponde ao seu risco. A cadência de revisão impede que os controles decaiam. Juntos, tornam a matriz de ownership aplicável, não aspiracional.

O princípio de design: construa uma vez, herde a aplicação

Os três hubs compartilham um princípio de design. Cada categoria de artefato, seja uma API, uma extensão, um servidor MCP ou um agente Copilot, recebe seu próprio contrato de governança, mas todo contrato é aplicado pelo mesmo plano de controle. Os artefatos mudam. O contrato não. É por isso que a plataforma não é uma camada de compliance colada sobre um ambiente de desenvolvimento existente. Ela é a própria plataforma.

O retorno é a herança. Construa o plano de controle uma vez, e toda nova categoria de artefato que surgir depois dos servidores MCP e dos agentes Copilot herda automaticamente a mesma superfície de aplicação. Quando a próxima categoria chega, o trabalho é escrever seu contrato de governança e roteá-la pelos portões existentes, não montar um novo projeto de governança. O IDE segue dentro do perímetro, mas nada chega a ele sem vetting. Esse é o ponto todo: governança que escala por herança, e não por repetição.

> DEFINIÇÃO
>
> Construa uma vez, herde a aplicação. Cada categoria de artefato recebe seu próprio contrato de governança; todos os contratos passam por um único plano de controle. Os artefatos mudam, o contrato não, e toda nova categoria herda a mesma superfície de aplicação sem um novo projeto de governança.

Referências

Fontes primárias para os cenários de deploy, a sequência do Dia 0 ao Dia 2 e os controles de governança em que este capítulo se apoia.

- What is Azure API Center: centralized inventory and governance, o inventário autoritativo provisionado no Dia 0.
- Link an Azure API Center to Azure API Management, o passo do Dia 1 que faz convergir a verdade do gateway e a do inventário.
- Enterprise extension management in Visual Studio Code, a base para distribuir o VS Code curado via Group Policy e Intune.
- Use MCP servers in Visual Studio Code, o modelo de confiança documentado por servidor antes de a frota consumi-lo.
- Analyze APIs in your API center: Spectral linting and analysis, o portão do Dia 2 que bloqueia APIs não conformes no registro.
- How to find shadow APIs with Dev Proxy and API Center, detecção contínua de shadow APIs em estado estável.

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