GOBERNANZA DE APIS Y TOOLING ENTERPRISE · PLAYBOOK

Un plano de control gobierna cada API, extensión y agente que llega a la estación de trabajo.

Los entornos de desarrollo empresariales acumulan riesgo de seguridad, cumplimiento y productividad más rápido de lo que cualquier equipo puede gobernar. Este playbook construye un plano de control sobre tres hubs, Azure API Center para APIs, un VS Code Private Marketplace para extensiones y gobernanza Copilot para servidores MCP y agentes, para que nada llegue a una estación de trabajo de desarrollador sin revisión. Los artefactos cambian. El contrato no.

AUTORA

Paula Silva

CARGO

AI-Native Software Engineer

CONTACTO

in/paulanunes

TIEMPO DE LECTURA

~ 45 minutos, de punta a punta

3	6	6
HUBS DE GOBERNANZA	CAPÍTULOS	PERSONAS DE PLATAFORMA

30-50%

SHADOW APIS ENCONTRADAS

CAPÍTULOS

Chapter 00

Dispersión sin plano de control

Por qué los entornos empresariales acumulan riesgo de seguridad, cumplimiento y productividad al mismo tiempo, cómo las shadow APIs y las extensiones sin revisión llegan a la estación de trabajo, la tesis del plano de control único y las seis personas de plataforma para quienes se escribió esta guía.

Concepto →

Chapter 01

Tres hubs, una estación de trabajo

El plano de control integrado y la capa de política compartida, el papel de cada hub, los conceptos que separan API Center de API Management, MCP y AllowedExtensions, y cómo los hubs convergen en VS Code con GitHub Copilot.

Arquitectura →

Chapter 02

Azure API Center como fuente de la verdad

La línea entre API Center y API Management, registro por portal, CLI y CI/CD, metadata obligatoria para cumplimiento, linting Spectral en la puerta, descubrimiento de shadow APIs con Dev Proxy y el flujo en VS Code.

Inventario de APIs →

Chapter 03

El Private Marketplace y la política de extensiones

Un registro stateless para extensiones internas, públicas rehospedadas y air-gapped, AllowedExtensions aplicada por Group Policy e Intune, el bootstrap install y la caída de footprint de cerca de 150 a aproximadamente 12 revisadas.

Marketplace →

Chapter 04

Gobernando MCP, agentes y artefactos de Copilot

Aprobación y distribución de servidores MCP, agentes custom y archivos de prompt, elección de modelo en el selector de GitHub Copilot, confirmaciones del agent mode y la arquitectura de seguridad que cubre cadena de suministro, prompt injection y auditoría.

AI Tooling →

Chapter 05

Del Día 0 al Día 2, y la gobernanza

Los tres escenarios de despliegue, la secuencia del Día 0 al Día 2 que levanta y mantiene la plataforma, la matriz de ownership con caminos de aprobación y cadencia de revisión, y el principio de herencia de construir una vez.

Operaciones →

CONSEJOS DE LECTURA

> **ATAJOS DE TECLADO**
>
> Presiona `/` para buscar. Usa `j` y `k` para moverte entre capítulos.
> Presiona `t` para cambiar tema. Presiona `g` para ir al inicio.

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 LA DISPERSIÓN

Dispersión sin plano de control: tres riesgos, una tesis, seis personas.

Los entornos de desarrollo empresariales acumulan riesgo más rápido de lo que cualquier equipo puede gobernar. Las APIs se multiplican sin inventario central. Las extensiones, los servidores MCP y los agentes Copilot llegan a las estaciones de trabajo de desarrollo sin revisión. Los tres riesgos que se derivan, seguridad, cumplimiento y productividad, no son independientes, y la mayoría de las organizaciones no tienen una superficie única desde la cual abordar los tres al mismo tiempo. Este capítulo nombra la dispersión, enuncia la tesis que la corrige e identifica las seis personas de plataforma para quienes se escribió esta guía.

Tres riesgos convergentes que no comparten superficie de control

Los entornos empresariales acumulan riesgo en tres categorías que se componen, y la mayoría de las organizaciones no tienen una superficie única desde la cual abordar las tres al mismo tiempo. La seguridad es el primer punto de presión. Las extensiones y servidores MCP instalados ad hoc, fuera de cualquier proceso de aprobación, pueden exfiltrar código o credenciales. El IDE se vuelve el blanco fácil. Está dentro del perímetro, tiene acceso profundo al sistema de archivos y a la red, y rara vez está cubierto por el mismo modelo de amenaza que la infraestructura de producción. El cumplimiento es el segundo. Datos sensibles llegan a las APIs de modelo de IA sin autorización, y los equipos de auditoría no pueden probar qué salió del entorno, cuándo salió ni a quién fue enviado. A medida que el desarrollo asistido por IA se vuelve estándar, y que esas llamadas de modelo se facturan a través de GitHub AI Credits, la brecha de cumplimiento se amplía más rápido de lo que la gobernanza manual puede cerrarla.

La productividad es el tercer riesgo, y con frecuencia el menos intuitivo. Cuando el desarrollador no descubre lo que ya existe, reconstruye. Usa endpoints deprecados.

Quema horas configurando tooling que ya existe en algún otro lugar de la organización. El costo de la duplicación crece silenciosamente. Los tres riesgos no son independientes. El tooling sin revisión impulsa el uso de shadow APIs. El uso de shadow APIs produce brechas en el flujo de datos. Las brechas en el flujo de datos generan exposición de cumplimiento. Un plano de control que aborda solo una categoría deja las otras dos abiertas, y por eso la corrección tiene que ser estructural, no un conjunto de soluciones puntuales añadidas sobre el entorno después del hecho.

Shadow APIs, extensiones sin revisión, servidores MCP y agentes

La dispersión tiene una forma específica. Las APIs proliferan sin inventario central, y las shadow APIs nacen porque los equipos no pueden ver lo que ya existe. Endpoints no documentados corren en producción, llamados por servicios que ningún registro conoce. En la práctica, una primera pasada de descubrimiento contra un gateway activo suele revelar entre treinta y cincuenta por ciento de las APIs de producción que no aparecen en ningún registro, y la mayoría de los equipos subestima esa proporción a la mitad hasta correr el descubrimiento. Esa brecha es la medida más nítida de cuánto se ha alejado el inventario de la realidad, y es la deuda que toda decisión de gobernanza posterior hereda.

El mismo patrón se repite por encima de la capa de API. Las extensiones VS Code se instalan sin revisión, y una sola base de desarrolladores suele extenderse a unas 150 extensiones distintas antes de que exista cualquier política. Los agentes de IA y los plugins Copilot envían datos a servicios externos sin supervisión. Los servidores MCP se configuran ad hoc, un desarrollador a la vez, cada uno un nuevo punto de integración con sus propias suposiciones de confianza. Los agentes custom definidos en archivos `.github/agents/` y las instrucciones en `.github/copilot-instructions.md` moldean cómo se comporta GitHub Copilot en agent mode, así que un archivo de instrucciones sin revisar es una brecha de gobernanza, no solo una comodidad. Cada uno de estos tipos de artefacto es un canal hacia la estación de trabajo, y cada canal es no gobernado por defecto.

La tesis: nada llega a una estación de trabajo de desarrollador sin revisión

La corrección es una regla, y cabe en una frase: nada llega a una estación de trabajo de desarrollador sin pasar por un catálogo gobernado. Tres hubs la aplican. Azure API Center es el inventario autoritativo para cada API de la organización, con linting Spectral en el momento del registro y descubrimiento vía Dev Proxy para las shadow APIs que ya circulan. El VS Code Private Marketplace es el catálogo curado para cada extensión, internas y públicas rehospedadas por igual. La gobernanza Copilot cubre cada servidor MCP, agente custom y archivo de prompt. Tres hubs, un plano de control.

Los tres hubs convergen en una única estación de trabajo de desarrollo, VS Code con GitHub Copilot, protegida por una capa de política compartida que aplica AllowedExtensions, RBAC, identidad y log de auditoría. GitHub Copilot en agent mode lee instrucciones de `.github/copilot-instructions.md` y llama herramientas vía MCP, así que la misma capa de política que gobierna extensiones y APIs también gobierna cómo se comporta el agente. El catálogo no es una puerta que frena a los desarrolladores. Es el camino rápido, porque el artefacto revisado es el que ya está instalado y ya es confiable desde el primer día.

Un plano de control, no tres herramientas

La idea no es tres herramientas separadas pegadas una a la otra. Cada categoría de artefacto, APIs, extensiones y tooling de IA, recibe su propio contrato de gobernanza, pero todo contrato se aplica a través del mismo plano de control. Constrúyelo una vez, y la próxima categoría de artefacto, lo que venga después de los servidores MCP y los agentes Copilot, hereda automáticamente la misma superficie de aplicación. Cuando los planos están conectados, cada superficie hacia el desarrollador produce un evento loggeado, y esa traza de auditoría es la diferencia entre un piloto en industria regulada que escala y uno que es revocado en la próxima auditoría externa.

DEFINICIÓN

Nada llega a una estación de trabajo de desarrollador sin pasar por un catálogo gobernado. Azure API Center gobierna cada API, el VS Code Private Marketplace gobierna cada extensión, y la gobernanza Copilot cubre cada servidor MCP, agente y archivo de prompt. Los artefactos cambian. El contrato no.

Para quién es esta guía: seis personas de plataforma

Esta guía está escrita para los equipos de plataforma que son dueños del plano de control, no para los desarrolladores que lo consumen. Seis personas se repiten entre los despliegues empresariales. El platform engineer construye y opera los tres hubs. El IT administrator aplica la política de extensiones vía Group Policy en Windows e Intune en macOS y dispositivos gestionados. El security engineer es dueño del modelo de amenaza, de los controles de supply chain y de la postura de auditoría. El developer enablement lead hace del camino gobernado el camino fácil, de modo que el catálogo curado también sea el más rápido. El enterprise architect mantiene los tres hubs coherentes con la plataforma más amplia. El API program manager trata el inventario como un producto, con dueños, clasificaciones y un ciclo de vida.

Cada persona lee la guía de forma distinta, y los capítulos que siguen están estructurados para eso. Los platform engineers suelen empezar por la arquitectura y la secuencia operativa. Los security engineers van directo al modelo de amenaza y al escenario de industria regulada. Los enablement leads se preocupan más por el camino de onboarding del desarrollador. La guía construye el plano de control un hub a la vez, luego conecta los hubs, para que cualquier persona pueda entrar por la capa que es suya y aun así ver cómo se enlaza con el resto de la plataforma.

Referencias

Fuentes primarias para los patrones de gobernanza de inventario, marketplace y tooling de IA de este capítulo.

- Azure API Center, inventario y gobernanza centralizados, el registro autoritativo para cada API, versión, entorno y deployment.
- Analizar APIs en tu API center, linting y análisis con Spectral, cómo las reglas de estándar de diseño bloquean APIs no conformes en el momento del registro.
- Encontrar shadow APIs con Dev Proxy y API Center, trayendo a la superficie APIs no documentadas que los desarrolladores ya llaman antes de que se vuelvan deuda en producción.
- Vincular un Azure API Center con Azure API Management, cómo la verdad del gateway y la verdad del inventario convergen en un único registro.
- Gestión corporativa de extensiones en Visual Studio Code, política AllowedExtensions vía Group Policy e Intune.
- Usar servidores MCP en Visual Studio Code, el modelo de instalación y distribución para servidores MCP gobernados.

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 LA ARQUITECTURA

Tres hubs, una estación de trabajo: el plano de control integrado.

Tres hubs convergen en una única estación de trabajo de desarrollo. Azure API Center gobierna las APIs, un marketplace privado gobierna las extensiones de VS Code y la gobernanza Copilot gobierna los artefactos de IA. Una capa de política compartida abarca los tres, y nada llega a la estación de trabajo sin pasar por ella. Este capítulo mapea la arquitectura: el plano de control, el papel de cada hub, los conceptos que los separan y cómo se encuentran dentro de VS Code con GitHub Copilot.

El plano de control integrado y la capa de política compartida

La arquitectura parte de una única estación de trabajo: VS Code (Desktop) ejecutando GitHub Copilot. Todo lo demás existe para gobernar lo que llega a esa superficie. El desarrollo empresarial acumula riesgo en tres frentes al mismo tiempo: extensiones y servidores MCP sin revisión que pueden exfiltrar código, datos sensibles que fluyen a las APIs de modelo sin autorización, y trabajo duplicado cuando el dev no encuentra lo que ya existe. Un plano de control que cierra solo uno de esos ciclos deja los otros dos abiertos. La arquitectura integrada cierra los tres gobernando tres categorías de artefacto a través de una única superficie de aplicación.

La capa de política compartida es esa superficie. Aplica AllowedExtensions, RBAC, identidad y log de auditoría en todos los hubs. AllowedExtensions es una política enterprise de VS Code que controla qué extensiones pueden instalarse y ejecutarse. El RBAC define quién puede leer y registrar artefactos. La propagación de identidad asocia un usuario real a cada acción, de modo que los logs de auditoría siguen siendo atribuibles. La regla es simple y estricta: nada llega a la estación de trabajo del desarrollador sin pasar por la revisión. La política no cambia cuando cambia el

tipo de artefacto. Las APIs, las extensiones y las herramientas de IA reciben cada una su propio contrato de gobernanza, y el mismo plano de control los aplica todos.

> DEFINICIÓN
>
> La regla ancla: nada llega a la estación de trabajo del desarrollador sin pasar por la revisión. Cada categoría de artefacto recibe su propio contrato de gobernanza, pero cada contrato se aplica a través del mismo plano de control. Los artefactos cambian. El contrato no.

Papeles de los hubs: Azure API Center, Private Marketplace, gobernanza Copilot

Cada hub es dueño de una categoría de artefacto. El Hub 01 es Azure API Center, el inventario autoritativo para cada API de la organización. Cada versión, cada definición, cada entorno y deployment se registra allí. Los pipelines de CLI y CI/CD alimentan el registro automáticamente, así que el catálogo refleja lo que los equipos realmente entregan. Los campos de metadata personalizada capturan la propiedad y la clasificación, lo que convierte al catálogo a la vez en una superficie de descubrimiento y en un registro de gobernanza. El linting Spectral se ejecuta en el momento del registro y bloquea APIs que violan estándares de diseño de la organización antes del deploy. El descubrimiento vía Dev Proxy detecta APIs no documentadas que los devs ya están llamando, de modo que las shadow APIs entran al catálogo antes de endurecerse como deuda de producción.

Marketplace y gobernanza Copilot

El Hub 02 es el VS Code Private Marketplace, un catálogo curado de extensiones bajo control de la propia empresa. El diseño es intencionalmente sin estado: un contenedor con object storage de respaldo, desplegable en cualquier región, que sirve extensiones internas y extensiones públicas rehospedadas. Los despliegues air-gap mantienen a los devs productivos sin ningún egress al marketplace público. La aplicación de políticas usa Group Policy en Windows e Intune en macOS y dispositivos gestionados. Un bootstrap install asegura que cada desarrollador abra un VS Code curado desde el primer día, no una instalación bruta y no gestionada.

El Hub 03 es la gobernanza Copilot, que aplica el mismo principio a los artefactos de IA. Los servidores MCP se registran en `.vscode/mcp.json` y se distribuyen a través del marketplace, cada uno con un modelo de confianza documentado. Las extensiones Copilot se revisan por flujo de datos y se agregan a la allowlist a nivel de la organización GitHub Enterprise. Los agentes personalizados viven en archivos `.github/agents/`, versionados junto al código que gobiernan y revisados en el mismo pipeline de pull request. Los prompts y skills se curan en un repositorio compartido con workflow de aprobación antes de llegar a la flota de desarrollo.

Conceptos clave: API Center vs API Management, MCP, AllowedExtensions

Tres conceptos cargan la mayor parte del peso, y el primero es la diferencia entre Azure API Center y Azure API Management. Son servicios separados con trabajos separados. API Management es el gateway de runtime: se sitúa frente al tráfico vivo, aplica políticas y enruta requests. API Center es el inventario y el registro de gobernanza: cataloga cada API sin importar dónde se ejecute, incluidas las APIs que nunca pasan por el gateway. Los dos se conectan, no se solapan. API Management alimenta a API Center upstream automáticamente, así que la verdad del gateway y la del inventario convergen en un único registro. Un equipo puede registrar una API en API Center antes de que exista cualquier gateway, y luego vincularla a API Management cuando entre en producción.

MCP y AllowedExtensions

El segundo concepto es el Model Context Protocol. El MCP es un estándar abierto que permite a un agente llamar herramientas y fuentes de datos externas a través de una interfaz uniforme. En esta arquitectura, los servidores MCP son la unidad de capacidad de IA: un servidor expone un conjunto de herramientas, GitHub Copilot en agent mode lo consulta bajo demanda, y el modelo de confianza de cada servidor se documenta antes del deploy. Registrar los servidores en `.vscode/mcp.json` y distribuirlos a través del marketplace convierte una integración ad hoc en un artefacto gobernado.

El tercer concepto es AllowedExtensions, la política enterprise de VS Code que define qué extensiones puede cargar una instalación gestionada. Es el mecanismo

detrás del marketplace privado: la política apunta el editor al catálogo interno y bloquea cualquier cosa fuera de él. Aplicado mediante Group Policy o Intune, AllowedExtensions es lo que hace que un marketplace curado sea obligatorio y no solo recomendado.

Cómo los hubs convergen en VS Code con GitHub Copilot

Los tres hubs resuelven al mismo lugar: el editor en el que trabaja el desarrollador. API Center es accesible desde dentro de VS Code, así que el dev descubre y registra APIs sin salir del IDE. El marketplace privado es desde donde se instalan las extensiones. La gobernanza Copilot decide qué agentes, servidores MCP e instrucciones están activos en la sesión. La estación de trabajo es donde la gobernanza se vuelve visible para quien hace el trabajo, y donde ayuda o estorba.

GitHub Copilot es la superficie que lo une todo. Al inicio de la sesión lee el `.github/copilot-instructions.md`, así que las convenciones del repositorio viajan junto al código. En agent mode consulta los servidores MCP que la gobernanza Copilot aprobó. El selector de modelos deja que el dev elija el modelo para la tarea, Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro o un modelo GPT-5.x, mientras la organización gobierna qué modelos y agentes están disponibles. El uso se factura en GitHub AI Credits, donde un crédito equivale a un centavo, así que la actividad de los agentes lleva un costo que la plataforma puede medir. El retorno operacional aparece en el Día 2: el linting bloquea APIs no conformes en la puerta, Dev Proxy detecta shadow APIs antes de producción, y los logs de auditoría dan a los equipos de cumplimiento visibilidad continua. Construye la superficie de aplicación una vez, y cada nueva categoría de artefacto la hereda.

Referencias

Fuentes primarias para la arquitectura de los tres hubs, la capa de política y los conceptos clave.

- Azure API Center overview, el hub de inventario autoritativo para cada API.

- Azure API Center key concepts, la distinción entre API Center y API Management.
- Link an Azure API Center to Azure API Management, cómo el gateway alimenta el inventario upstream.
- Enterprise extension management in Visual Studio Code, AllowedExtensions e instalaciones gestionadas.
- Use MCP servers in Visual Studio Code, registro de servidores MCP para GitHub Copilot en agent mode.
- Model Context Protocol Specification, el estándar abierto detrás del hub de capacidad de IA.
- GitHub Copilot custom instructions, cómo el .github/copilot-instructions.md llega a la sesión.
- Using GitHub Copilot with an AI model of your choice, el selector de modelos en la estación de trabajo.

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 INVENTARIO DE APIS

Azure API Center como fuente de la verdad: un registro para cada API.

Una plataforma de gobernanza es tan honesta como su inventario. Si un equipo no puede decir qué APIs existen, qué versión es canónica y cuáles tocan datos sensibles, todo control por encima de esa brecha es conjetura. Este capítulo pone Azure API Center en la base del stack: cómo se diferencia de API Management, cómo se registran y clasifican las APIs, cómo el linting y el descubrimiento de shadow APIs mantienen el registro honesto, y cómo los desarrolladores llegan a todo eso desde dentro de VS Code.

Trazando la línea entre API Center y API Management

Azure API Center y Azure API Management resuelven problemas distintos, y la plataforma se rompe cuando un equipo los trata como un solo producto. Azure API Center es el registro único de cada API de la organización. Graba qué APIs existen, qué versión es canónica, si una definición cumple con la guía de estilo y quién es dueño de cada entrada. Guarda metadata, versiones, definiciones OpenAPI y entornos de deployment a lo largo de todo el ciclo de vida, desde el diseño inicial hasta la deprecación. Azure API Management es la capa de runtime. Es el gateway que publica, protege, aplica rate limit y monitorea el tráfico en vivo del subconjunto de APIs que gestiona.

La relación va en una dirección y luego vuelve. El API Center conoce todas las APIs. El API Management gobierna el tráfico de las que expone. Los dos se conectan por sincronización: vincula una instancia de Azure API Management y todo su catálogo fluye al API Center automáticamente, de modo que la verdad del gateway y la del inventario convergen en un único registro, sin reingreso manual. Una forma práctica de explicarlo a los stakeholders es decir que el API Management es el agente de tránsito en el cruce, y el API Center es la oficina de planeación que sabe que cada

calle existe. La línea también importa para el costo. Al vincular una instancia de API Management en los tiers Standard, Standard v2, Premium o Premium v2, el plan Standard del API Center llega sin costo adicional mientras al menos una instancia calificada siga vinculada.

Registro por portal, CLI y CI/CD, y metadata para cumplimiento

Un registro solo gana confianza cuando cada API llega a él de la misma forma cada vez. El API Center admite tres caminos de registro que reflejan cómo trabajan los equipos de verdad. El portal de Azure hace el registro manual: agrega la API, define el tipo (REST, GraphQL, gRPC, SOAP, Webhook), elige una etapa de ciclo de vida y sube la definición OpenAPI. La CLI de Azure scripta los mismos pasos con az apic api create, az apic api version create y az apic api definition import-specification, lo que hace el registro repetible y revisable. El CI/CD cierra la brecha. Un workflow de GitHub Actions disparado por cambios en archivos openapi registra una nueva versión en cada merge a main, y el catálogo nunca queda detrás del código.

La metadata convierte una lista en un inventario gobernado

El registro graba que una API existe. La metadata personalizada graba lo que cumplimiento y seguridad necesitan saber sobre ella. Define las propiedades una vez con az apic metadata create y exígelas en cada entrada: un dueño del equipo, un booleano para indicar si la API procesa PII, un enum de dominio de negocio. Marca las propiedades como obligatorias y el catálogo deja de aceptar APIs huérfanas, sin dueño y sin clasificación de datos. Eso es lo que separa una superficie de descubrimiento de un registro de gobernanza. Cuando un auditor pregunta qué APIs de producción tocan PII, la respuesta es un filtro de metadata, no una encuesta.

Linting Spectral en la puerta, descubrimiento de shadow APIs con Dev Proxy

Dos controles mantienen el registro honesto: uno bloquea specs malos en la entrada, el otro encuentra las APIs que nunca pidieron permiso. El API Center corre linting

integrado movido por el motor open source Spectral. Sube la guía de estilo de tu organización como un ruleset Spectral, y cada definición OpenAPI se valida contra ella en el momento del registro. Las definiciones que rompen reglas de nomenclatura, versionado o seguridad se marcan antes de llegar a producción, no se atrapan en la revisión tres semanas después. Las reglas son las mismas en todas partes, así que el diseño de API deja de depender de quién revisó el pull request.

Las shadow APIs son las llamadas que una aplicación ya hace y que no aparecen en ningún catálogo. Son deuda técnica y exposición de cumplimiento al mismo tiempo. Microsoft Dev Proxy intercepta el tráfico HTTP de la aplicación, graba cada llamada y compara los endpoints observados con el inventario del API Center. El reporte lista cada API llamada pero nunca registrada, con pasos de registro sugeridos. En la práctica, los equipos que corren el descubrimiento por primera vez encuentran que del 30 al 50 por ciento de las APIs en producción faltaban en cualquier registro. Traer esas APIs al API Center antes de que se acumulen es la diferencia entre un inventario que refleja la realidad y uno que documenta intenciones.

El flujo en VS Code, del browse al grounding de Copilot

El registro solo sirve si los desarrolladores llegan a él sin salir del editor. La extensión de Azure API Center para VS Code trae el inventario completo a la Activity Bar. Los desarrolladores navegan las instancias por suscripción, abren cualquier definición OpenAPI con syntax highlighting y filtran por metadata con una query como domain:payments pii:true lifecycle:production. Registran nuevas APIs con clic derecho sobre un spec, lo que mantiene el registro como acto de diseño, no como limpieza post-deployment. Generan un cliente tipado en TypeScript, Python, C#, Java o Go a partir de cualquier definición registrada, para que el código del consumidor coincida con la versión actual en lugar de una conjetura escrita a mano. El mismo ruleset Spectral corre inline mientras escriben, exponiendo violaciones en el panel Problems antes de que el spec se suba.

Aquí es donde GitHub Copilot gana su lugar. Con la extensión @azure de Copilot habilitada, el agent mode ancla sus respuestas en el inventario vivo del API Center en lugar de adivinar. Pregunta qué APIs manejan autenticación de clientes, o cuáles procesan PII, y Copilot devuelve respuestas contextuales desde el registro, no desde

los datos de entrenamiento. Apúntalo a un controller y redacta una spec OpenAPI que cumple con el estilo registrado, lista para registrar mediante la extensión. Expón el inventario por el Model Context Protocol y cualquier agente del picker, sea Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro o GPT-5.x, trabaja desde el mismo contexto autoritativo. Registra las reglas de grounding en .github/copilot-instructions.md para que el agente de cada desarrollador parta de la misma fuente de la verdad. La facturación de GitHub Copilot corre sobre GitHub AI Credits, donde un crédito equivale a un centavo de dólar, así que las consultas ancladas que evitan rederivar el catálogo en cada sesión salen más baratas además de más precisas.

> DEFINICIÓN
>
> La regla es simple: una API es real cuando está en API Center, no cuando está en producción. El registro es un acto de diseño, la metadata es obligatoria y no opcional, el linting corre en la puerta, y Dev Proxy encuentra lo que se escapó. Todo lo que el desarrollador y el agente hacen pasa por un único registro.

Referencias

Fuentes primarias en Microsoft Learn para el API Center como hub de inventario, sus controles de registro y gobernanza, y el flujo en VS Code.

- Microsoft Learn, What is Azure API Center, el hub central de inventario y gobernanza para cada API de la organización.
- Microsoft Learn, Synchronize APIs from Azure API Management, cómo la verdad del gateway fluye al inventario automáticamente.
- Microsoft Learn, Register APIs with GitHub Actions, registro por CI/CD que mantiene el catálogo al día en cada merge.
- Microsoft Learn, Enable API analysis and linting, reglas Spectral validando cada definición en el momento del registro.
- Microsoft Learn, Discover shadow APIs with Dev Proxy, encontrando las APIs que corren en producción pero no aparecen en ningún registro.
- Microsoft Learn, Build and register APIs with the VS Code extension, el flujo de browse, registro y generación de cliente dentro del 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

El Private Marketplace y la política de extensiones: un único canal hasta el editor.

Un marketplace curado es el segundo plano de control. El primero gobierna APIs. Este gobierna lo que corre dentro del editor. Cada extensión, cada servidor MCP y cada artefacto de GitHub Copilot llega al desarrollador por un único canal que el equipo de plataforma controla. Este capítulo describe ese canal: un registro stateless, la policy AllowedExtensions que lo aplica, el bootstrap install que lo siembra y la reducción de footprint que produce.

Un registro stateless: internas, públicas rehospedadas y air-gapped

El Private Marketplace es un catálogo curado de extensiones que la empresa controla. Su arquitectura es stateless por diseño. El registro corre como un container respaldado por object storage, y puede desplegarse en cualquier región que la organización necesite. Sirve dos tipos de extensión. El primero es interno: extensiones que la organización construye para sus propios flujos, publicadas directo en el catálogo. El segundo es público rehospedado: extensiones del marketplace público de VS Code, descargadas, revisadas y republicadas en almacenamiento interno para que el editor nunca tenga que alcanzar la galería pública.

El diseño stateless es lo que hace funcionar el escenario air-gapped. En una red restringida sin egress público, el marketplace mantiene en almacenamiento interno cada extensión que un desarrollador necesita, y el editor apunta al endpoint interno en lugar del público. El mismo registro sirve a la empresa conectada a la nube y al banco regulado. Solo cambia la frontera de red. Este es el Hub 02 en la arquitectura de tres hubs, y queda bajo la misma capa de política que Azure API Center y la gobernanza Copilot. Nada se instala en el editor sin pasar por él.

AllowedExtensions aplicada mediante Group Policy e Intune

Un catálogo es una sugerencia hasta que una policy lo vuelve la única opción. AllowedExtensions es esa policy. Es una configuración enterprise de VS Code que nombra las extensiones que un editor gestionado tiene permitido instalar, y bloquea todo lo demás. En Windows el equipo de plataforma la distribuye por Group Policy. En macOS y otros dispositivos gestionados va por Intune. El mecanismo cambia según el sistema operativo. El resultado es idéntico: el editor del desarrollador puede instalar lo que el catálogo permite y nada más.

La policy no es un candado que atrapa a la gente. Una extensión fuera de la allowlist sigue siendo instalable a pedido, y el pedido deja una aprobación trazable. Esa distinción importa para la postura de cumplimiento. En un deploy regulado, cada extensión en cada workstation vuelve a un registro de aprobación, y un auditor puede leer la lista sin pedirle al equipo de plataforma que la reconstruya. AllowedExtensions también gobierna la superficie de IA. Una extensión de GitHub Copilot, o cualquier extensión que cargue un servidor MCP, está sujeta al mismo portón que un resaltador de sintaxis. El catálogo curado es donde vive el tooling de IA aprobado, no un canal separado.

Bootstrap install y el catálogo enterprise recomendado

La policy decide qué está permitido. El bootstrap install decide con qué empieza un desarrollador. Un bootstrap install es un setup de primera ejecución por script que entrega a cada desarrollador un VS Code curado desde el primer día, no un editor crudo que configura a mano. Apunta el editor al Private Marketplace, aplica la policy AllowedExtensions e instala el catálogo recomendado. Un recién llegado abre el editor y encuentra los linters, el soporte de lenguaje y las extensiones internas que el equipo ya acordó.

El catálogo recomendado es donde el lente de GitHub Copilot se afina. GitHub Copilot llega en ese install ya gobernado. Su selector de modelos viene precargado con los modelos que la organización aprobó, tomados del mismo conjunto que todo usuario de Copilot ve: Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y

la familia GPT-5.x. Su comportamiento se moldea con un .github/copilot-instructions.md versionado, y los servidores MCP que puede alcanzar son los registrados en .vscode/mcp.json y distribuidos por el mismo marketplace. El trabajo en agent mode se factura en GitHub AI Credits, donde un crédito es un centavo bajo el modelo vigente desde junio de 2026, y un install gobernado es lo que vuelve ese gasto atribuible a un equipo y a un repositorio, en lugar de una línea misteriosa en la factura.

Footprint de referencia: 150 extensiones a cerca de 12 revisadas

El número del footprint es el que se defiende en una revisión. Una base típica de desarrolladores acumula alrededor de 150 extensiones distintas entre todas las workstations, instaladas ad hoc con el tiempo sin revisión. Cuando AllowedExtensions aplica el catálogo curado, ese número baja a cerca de 12 extensiones revisadas más las internas que la organización publica. Esta es una reducción de referencia observada en la práctica, no una meta fijada por política. La revisión es lo que encoge la superficie, y la policy es lo que la mantiene ahí.

Un footprint menor es una superficie de ataque menor. Cada extensión instalada en el editor corre con el acceso a sistema de archivos y red del desarrollador, y por eso una extensión sin revisión es un camino plausible para exfiltración de código o credenciales. Recortar de 150 a 12 quita la mayor parte de esa exposición sin quitar capacidad, porque las 12 son las que los desarrolladores de verdad necesitaban. Las extensiones bloqueadas siguen disponibles a pedido, así que un desarrollador con una necesidad real abre un pedido y recibe una aprobación trazable. El default es curado. La excepción está documentada. Ese es todo el principio de diseño del marketplace, y es el mismo contrato que Azure API Center aplica a las APIs y la gobernanza Copilot aplica a los agentes.

DEFINICIÓN

El footprint de extensiones en una línea: de cerca de 150 extensiones ad hoc en una base de desarrolladores a unas 12 revisadas más las extensiones internas, aplicado por AllowedExtensions y reversible por una aprobación trazable. Curado por default, documentado por excepción, atribuible de punta a punta.

Referencias

Fuentes primarias para la arquitectura del Private Marketplace, la mecánica de la política de extensiones y la distribución de tooling de IA descrita en este capítulo.

- Visual Studio Code, Private Marketplace announcement, la referencia para el registro stateless que sirve extensiones internas y públicas rehospedadas.
- VS Code Private Marketplace, setup and rehosting readme, pasos de deploy y rehospedaje de extensiones públicas para redes air-gapped y conectadas.
- Visual Studio Code, Enterprise extension management, la configuración AllowedExtensions que vuelve el catálogo curado el único conjunto instalable.
- Visual Studio Code, Enterprise policies, cómo la policy se distribuye por Group Policy en Windows e Intune en dispositivos gestionados.
- Use MCP servers in Visual Studio Code, cómo los servidores MCP usan el mismo canal de distribución que las extensiones 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

Gobernando MCP, agentes y artefactos de Copilot: el contrato del tooling de IA.

Los servidores MCP, los agentes custom, los archivos de prompt y las instrucciones de Copilot son código que corre dentro de la estación de trabajo del desarrollador. Leen el sistema de archivos, llaman a APIs internas y actúan en nombre del desarrollador. Trátalos como artefactos de tooling, con el mismo contrato de aprobación, distribución y auditoría que cualquier extensión o cualquier API. Este capítulo define ese contrato para la capa de tooling de IA y cierra con la arquitectura de seguridad que la sostiene.

Servidores MCP: aprobación, distribución y credenciales

El Model Context Protocol conecta un agente de GitHub Copilot con herramientas y datos externos durante una conversación. Sin él, Copilot lee solo el código en el editor. Con él, un agente en agent mode puede consultar Jira, leer una base de datos, llamar a una API interna o buscar en Confluence. MCP usa un modelo cliente-servidor. VS Code es el cliente. Cada sistema expone un servidor MCP que declara sus capacidades como tools, resources y prompts. El agente decide qué tools invocar a partir del pedido que recibe. Los servidores corren localmente vía stdio o remotamente vía HTTP. Para herramientas organizacionales, los servidores remotos compartidos son el modelo preferido: una configuración, un único lugar para gestionar credenciales.

Un servidor MCP es un artefacto de tooling del desarrollador. Gobiérnalo con el mismo rigor que una extensión o una API. Un servidor que conecta a una base de datos interna es tan sensible como las credenciales de la base que guarda. Todo servidor pasa por una puerta de seguridad antes de llegar al catálogo aprobado: código fuente o informe del proveedor revisado, permisos al mínimo necesario,

read-only siempre que sea posible, servidores remotos en HTTPS con autenticación, y ninguna transmisión de contenido de solicitud a destinos no autorizados. Solo entonces el servidor se agrega al catálogo de la organización y queda habilitado para distribución.

Distribución y manejo de credenciales

Distribuye los servidores aprobados donde los desarrolladores ya trabajan. Un archivo `.vscode/mcp.json` versionado en el repositorio da a cada clon las mismas herramientas, sin configuración manual. Una configuración base entregada vía Intune o JAMF precarga los servidores aprobados por la organización en cada estación de trabajo gestionada, y los desarrolladores agregan servidores específicos del repositorio encima. Las credenciales nunca viven en `settings.json` ni en `mcp.json` como cadenas literales. Referéncialas como variables de entorno y obtenlas de un gestor de secretos como Azure Key Vault al inicio de la sesión. La configuración declara qué secreto necesita; el secreto en sí permanece en el vault. La exposición de credenciales y el prompt injection son los dos riesgos nombrados del modelo de confianza de MCP, y ambos se abordan aquí.

Agentes custom, archivos de prompt e instrucciones

GitHub Copilot tiene varios tipos de artefacto, y cada uno lo extiende de forma distinta. Un agente custom es un archivo Markdown en `.github/agents/`, seleccionado desde el selector de agentes en Copilot Chat. Codifica el conocimiento de dominio del equipo, las convenciones y el acceso a herramientas con alcance, directo en el repositorio. Un archivo de prompt es una plantilla reutilizable `.prompt.md`, invocada con un slash command para tareas complejas recurrentes. Las instrucciones de Copilot viven en `.github/copilot-instructions.md` y se aplican automáticamente a cada sesión de Chat en el repositorio. Como los tres son archivos, quedan versionados junto al código que gobiernan y se revisan en el mismo pipeline de pull request.

Coloca un agente en el directorio `.github/agents/` del repositorio `.github` de la organización y queda disponible en todos los repositorios, sin configuración por repositorio. Lo mismo vale para archivos de prompt e instrucciones a nivel de

organización. Ese alcance es justamente lo que exige un contrato de revisión de los artefactos. Los agentes custom se revisan por el equipo de developer enablement antes del merge. Los archivos de prompt que llaman a comandos de terminal requieren revisión de seguridad. Un inventario central de agentes, prompts e instrucciones evita la proliferación descontrolada, que es la versión de tooling de IA de la dispersión de extensiones.

Las instrucciones como superficie de fundamentación

El archivo de instrucciones es donde el repositorio le dice al agente cómo funciona realmente el código: los estándares de API, la stack de tecnología, las convenciones de nombres y de errores. Es también donde viven las barreras de seguridad. Instruye al agente a tratar el texto devuelto por herramientas externas como dato no confiable, a mostrar el comando completo antes de modificar infraestructura y a nunca hacer push a un remoto sin confirmación explícita. Las instrucciones son fundamentación, no decoración. Un agente que las lee produce código que pertenece al repositorio, en vez de código que solo compila.

Elección de modelo y confirmaciones del agent mode

GitHub Copilot expone un selector de modelos. El desarrollador elige el modelo por tarea a partir del conjunto disponible: Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y la familia GPT-5.x. Un agente custom puede fijar un modelo por defecto en su definición, y el desarrollador puede sobreescribirlo en el selector. Ajusta el modelo al trabajo. Un refactor grande en muchos archivos recompensa un modelo más fuerte; una edición corta y bien acotada corre bien en uno más rápido. La facturación se mide en GitHub AI Credits, donde un crédito equivale a un centavo, en vigor desde el 1 de junio de 2026. La elección de modelo es, por lo tanto, una decisión de costo tanto como de calidad, y pertenece a la misma conversación de gobernanza que todo lo demás en esta capa.

El agent mode es donde el modelo deja de sugerir y empieza a actuar: edita archivos, corre comandos de terminal e invoca tools MCP. Esa autonomía necesita un freno. Configura VS Code para exigir confirmación del desarrollador antes de acciones sensibles: comandos de terminal, escrituras de archivo e invocaciones

MCP. Deja esas confirmaciones en always para operaciones de escritura. El prompt de confirmación es el punto donde un humano lee el plan y aprueba o rechaza cada paso. Es la diferencia entre un agente que ayuda y un agente que sorprende.

> DEFINICIÓN
>
> La regla del agent mode: la confirmación es obligatoria para toda escritura. Los comandos de terminal, las escrituras de archivo y las invocaciones MCP requieren, cada uno, una aprobación humana explícita. El desarrollador ve exactamente lo que el agente hará antes de que lo haga.

Arquitectura de seguridad: modelo de amenaza, cadena de suministro, auditoría

La capa de tooling de IA agrega superficies de ataque, y cada una necesita una mitigación nombrada. Una extensión maliciosa puede exfiltrar código: la policy AllowedExtensions y un marketplace privado con revisión de seguridad la bloquean. Una respuesta de tool MCP puede cargar un prompt injection: la confirmación para acciones de escritura y las barreras del agente lo contienen. Una extensión Copilot de terceros puede recibir contexto de código: una allow list a nivel de organización y una revisión de flujo de datos la gobiernan. Un agente custom puede ejecutar una acción destructiva: las políticas de confirmación y las instrucciones de barrera lo restringen. La seguridad de la cadena de suministro es defensa en profundidad. La policy bloquea extensiones no aprobadas, el marketplace privado rehospeda solo paquetes revisados, y una auditoría regular compara lo instalado con la lista aprobada.

El prompt injection es el riesgo característico de la capa agéntica. Ocurre cuando el dato devuelto por una tool, el cuerpo de un ticket de Jira o una página de Confluence, contiene instrucciones que manipulan al agente. Las mitigaciones son en capas: exigir confirmación antes de cada acción del agente, entrenar a los desarrolladores para leer el plan completo antes de aprobar, y agregar instrucciones de barrera que marcan toda salida de herramienta externa como no confiable. Por encima de los controles está la traza de auditoría. Cuando los planos se conectan, cada llamada de API, cada invocación MCP y cada acción de agente produce un

evento loggeado en Microsoft Purview o el SIEM elegido. La traza de auditoría es la diferencia entre un piloto en industria regulada que escala y uno que es revocado en la próxima auditoría externa.

> DEFINICIÓN
>
> Trata el texto devuelto por herramientas externas como dato no confiable. Un ticket de Jira, una página de Confluence o un registro de base de datos puede cargar instrucciones dirigidas al agente, y el agente no debe seguirlas. Las puertas de confirmación más las instrucciones de barrera en copilot-instructions.md son la defensa.

Referencias

Fuentes primarias para la gobernanza de MCP, los agentes custom, la elección de modelo y la postura de seguridad de Copilot.

- Model Context Protocol Specification, el estándar que conecta agentes de Copilot con herramientas y datos externos.
- Use MCP servers in Visual Studio Code, configuración y distribución vía mcp.json.
- GitHub Copilot Custom Agents, el formato de agente Markdown en .github/agents/.
- GitHub Copilot Custom Instructions, la superficie de fundamentación y barrera en copilot-instructions.md.
- Using GitHub Copilot with an AI model of your choice, elección de modelo en el selector de Copilot.
- GitHub Copilot Trust Center, privacidad de datos y postura de seguridad para 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 EJECUTAR Y GOBERNAR

Del Día 0 al Día 2, y el framework de gobernanza.

Una arquitectura vale lo que valen su rollout y su estado estable. Este capítulo cubre los tres escenarios de despliegue que soporta el plano de control, la secuencia del Día 0 al Día 2 que levanta y mantiene la plataforma, y el framework de gobernanza que define quién es dueño de qué y cómo cada artefacto llega a un desarrollador. El objetivo de diseño es una plataforma en la que una nueva categoría de artefacto entra sin un nuevo proyecto de gobernanza.

Tres escenarios de despliegue: integrado completo, air-gapped, regulado

La misma arquitectura soporta tres formatos de despliegue, y la elección depende de la postura de red y de la carga regulatoria, no de un diseño distinto. En el escenario integrado completo, la empresa está conectada a la nube. El diseño de API ocurre en VS Code, se registra en Azure API Center, se sincroniza con Azure API Management y se consume de vuelta en el IDE. El VS Code Private Marketplace y GitHub Copilot corren online. El agent mode lee el .github/copilot-instructions.md y llama a servidores MCP registrados en .vscode/mcp.json. Este es el baseline, y ejercita cada hub de extremo a extremo.

El escenario air-gapped elimina el egress de red pública. El marketplace rehospeda extensiones públicas en object storage interno, los servidores MCP corren en endpoints internos, y GitHub Copilot Enterprise alcanza la organización por private link. Los desarrolladores siguen productivos sin ningún camino al marketplace público. El escenario regulado, para finanzas y sector público, añade profundidad a la postura de cumplimiento en lugar de cambiar la topología: retención de log de auditoría, etiquetas de clasificación en los artefactos, binarios firmados y propagación de identidad por request. El auditor es independiente del equipo de plataforma. Cada acción es atribuible y cada artefacto es trazable.

DEFINICIÓN

Una arquitectura, tres posturas. Integrado completo ejercita cada hub online. Air-gapped elimina el egress y rehospeda internamente. Regulado añade retención, clasificación, artefactos firmados e identidad por request. La topología no cambia; los controles se aprietan.

Del Día 0 al Día 2: planificar, desplegar, operar

El Día 0 es planificación y standup de la plataforma. Azure API Center se aprovisiona como el inventario autoritativo. El VS Code Private Marketplace se despliega como un contenedor stateless con object storage detrás. Las políticas de gobernanza de GitHub Copilot se configuran. La identidad y el RBAC se conectan en los tres hubs antes de hacer onboarding a un solo desarrollador. El Día 0 también decide el presupuesto de AI Credits para el uso de Copilot: desde el 1 de junio de 2026, las solicitudes premium de GitHub Copilot se facturan en AI Credits, donde un crédito es un centavo de dólar, así que la elección en el model picker entre Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y la familia GPT-5.x es una decisión de costo que el equipo de plataforma planifica de antemano.

El Día 1 es la secuencia de despliegue y el onboarding de la flota. Los bootstrap installs distribuyen instancias de VS Code curadas vía Group Policy en Windows e Intune en dispositivos macOS gestionados, para que cada desarrollador abra un IDE gobernado el primer día, y no una instalación bruta. Los registros de APIs empiezan a fluir vía CI/CD hacia API Center. Azure API Management se vincula para que la verdad del gateway y la del inventario converjan en un único registro. Los modelos de confianza de los servidores MCP se documentan, uno por servidor, antes de que los servidores lleguen a la flota. Los agentes custom en .github/agents/ y las instrucciones en .github/copilot-instructions.md se versionan junto al código que gobiernan.

Día 2: estado estable y onboarding continuo

El Día 2 es operación en estado estable, y es donde la plataforma demuestra su valor. El linting Spectral corre en el momento del registro y bloquea APIs no

conformes en la puerta. Dev Proxy trae a la superficie las shadow APIs que los desarrolladores ya están llamando y las incorpora al catálogo antes de que se vuelvan deuda de producción. Los logs de auditoría dan a los equipos de cumplimiento visibilidad continua. El onboarding no se detiene en el Día 1: cada nueva API, extensión, servidor MCP y agente Copilot entra por las mismas puertas, así que la flota crece sin debilitar la gobernanza.

> DEFINICIÓN
>
> El Día 0 levanta API Center, el Private Marketplace, las políticas de Copilot, la identidad y el RBAC, y define el presupuesto de AI Credits. El Día 1 empuja el VS Code curado vía Group Policy e Intune, hace fluir los registros de APIs por CI/CD y documenta los modelos de confianza MCP. El Día 2 corre las puertas: el linting bloquea APIs no conformes, Dev Proxy captura shadow APIs, los logs de auditoría siguen continuos.

El framework de gobernanza: ownership, caminos de aprobación, cadencia de revisión

La gobernanza empieza con una matriz de ownership. Cada categoría de artefacto tiene un dueño nombrado y un aprobador definido. Las APIs son propiedad del equipo productor y se aprueban contra estándares de diseño organizacionales aplicados por Spectral. Las extensiones son propiedad del equipo de plataforma y se agregan a la allowlist por el marketplace. Los servidores MCP llevan un modelo de confianza documentado y un dueño responsable de los datos que pueden alcanzar. Las extensiones Copilot se revisan por flujo de datos y se agregan a la allowlist a nivel de la organización GitHub Enterprise. Los agentes custom son propiedad del equipo cuyo código tocan. El RBAC en API Center y los controles a nivel de organización en GitHub hacen la matriz aplicable en lugar de aspiracional.

Cada categoría de artefacto tiene su propio camino de aprobación, y el camino corresponde al riesgo del artefacto. Una API se aprueba cuando pasa el linting en el registro y lleva la metadata obligatoria de ownership y clasificación. Una extensión se aprueba cuando se revisa y se publica en el marketplace privado. Un servidor MCP se aprueba cuando su modelo de confianza está documentado y su endpoint

está vetado. Un agente Copilot o un archivo de instrucciones en .github/copilot-instructions.md se aprueba en el mismo pipeline de pull request del código que gobierna, revisado por las mismas personas, bajo las mismas branch protections. El camino de aprobación es code review, no una fila de tickets separada.

Cadencia de revisión

La cadencia de revisión impide que los controles se degraden. Las reglas de linting y los esquemas de clasificación se revisan en un ciclo fijo a medida que los estándares evolucionan. Los modelos de confianza de los servidores MCP se vuelven a revisar cuando el alcance de un servidor cambia. La allowlist se audita para que nada permanezca sin un dueño. La política del model picker se revisa a medida que nuevos modelos entran en el picker y a medida que llegan los datos de consumo de AI Credits, porque el costo de las solicitudes premium es una señal de gobernanza, no solo una línea de finanzas. La cadencia es la diferencia entre una gobernanza que se sostiene y una que se erosiona en silencio.

> DEFINICIÓN
>
> El ownership nombra a quién es responsable. Los caminos de aprobación enrutan cada artefacto por el control que corresponde a su riesgo. La cadencia de revisión impide que los controles se degraden. Juntos, hacen la matriz de ownership aplicable, no aspiracional.

El principio de diseño: constrúyelo una vez, hereda la aplicación

Los tres hubs comparten un principio de diseño. Cada categoría de artefacto, sea una API, una extensión, un servidor MCP o un agente Copilot, recibe su propio contrato de gobernanza, pero todo contrato se aplica a través del mismo plano de control. Los artefactos cambian. El contrato no. Por eso la plataforma no es una capa de cumplimiento pegada sobre un entorno de desarrollo existente. Es la plataforma misma.

El retorno es la herencia. Construye el plano de control una vez, y cada nueva categoría de artefacto que aparezca después de los servidores MCP y los agentes Copilot hereda automáticamente la misma superficie de aplicación. Cuando llega la siguiente categoría, el trabajo es escribir su contrato de gobernanza y enrutarla por las puertas existentes, no montar un nuevo proyecto de gobernanza. El IDE sigue dentro del perímetro, pero nada llega a él sin revisión. Ese es todo el punto: gobernanza que escala por herencia, no por repetición.

> DEFINICIÓN
>
> Constrúyelo una vez, hereda la aplicación. Cada categoría de artefacto recibe su propio contrato de gobernanza; todos los contratos pasan por un único plano de control. Los artefactos cambian, el contrato no, y cada nueva categoría hereda la misma superficie de aplicación sin un nuevo proyecto de gobernanza.

Referencias

Fuentes primarias para los escenarios de despliegue, la secuencia del Día 0 al Día 2 y los controles de gobernanza en los que se apoya este capítulo.

- What is Azure API Center: centralized inventory and governance, el inventario autoritativo aprovisionado en el Día 0.
- Link an Azure API Center to Azure API Management, el paso del Día 1 que hace converger la verdad del gateway y la del inventario.
- Enterprise extension management in Visual Studio Code, la base para distribuir el VS Code curado vía Group Policy e Intune.
- Use MCP servers in Visual Studio Code, el modelo de confianza documentado por servidor antes de que la flota lo consuma.
- Analyze APIs in your API center: Spectral linting and analysis, la puerta del Día 2 que bloquea APIs no conformes en el registro.
- How to find shadow APIs with Dev Proxy and API Center, detección continua de shadow APIs en estado estable.

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