CONSTRUYENDO AGENTES DE IA ENTERPRISE EN MICROSOFT · PLAYBOOK

Un agente percibe, decide y actúa, delimitado por política.

Los agentes de IA enterprise no son chatbots. Perciben, deciden y actúan, con autonomía delimitada por política. Este playbook mapea las tres plataformas Microsoft, Copilot Studio, GitHub Copilot y Azure AI Foundry, los cuatro tipos de agente y la arquitectura de seis capas, los guardrails de Human-in-the-Loop que escalan, dos casos en producción y una hoja de ruta de adopción de doce meses, todo visto a través de GitHub Copilot.

AUTORA

Paula Silva

CARGO

AI-Native Software Engineer

CONTACTO

in/paulanunes

TIEMPO DE LECTURA

~ 45 minutos, de punta a punta

3
PLATAFORMAS MICROSOFT

6
CAPÍTULOS

4
TIPOS DE AGENTE

5
FASES DE ADOPCIÓN

CAPÍTULOS

Chapter 00

Qué es un agente, y qué no es

La definición de trabajo, percibir, decidir, actuar, delimitado por política, chatbot versus agente, los cinco componentes que todo agente comparte y el stack de decisión de cuatro capas.

Fundamentos →

Chapter 01

Tres plataformas, un framework

Cómo elegir entre Copilot Studio, GitHub Copilot y Azure AI Foundry por público, nivel de habilidad, tiempo hasta el piloto y modelo de gobernanza.

Plataformas →

Chapter 02

Tipos y la arquitectura de seis capas

Los cuatro tipos de agente enterprise ajustados al trabajo, las seis capas desde la gobernanza hasta la superficie, herramientas MCP, elección de modelo y RAG del modo correcto.

Arquitectura →

Chapter 03

Gobernanza Human-in-the-Loop

Los cuatro cuadrantes de HITL por impacto y reversibilidad, tres patrones de implementación, y cómo reutilizar la gobernanza de Azure en lugar de controles paralelos.

Gobernanza →

Chapter 04

Casos en producción

Arrendamiento de vehículos en producción: cuatro agentes contra un plazo de multa de 15 días, tres agentes para el cumplimiento fiscal de NF-e, y el patrón ingerir, clasificar, actuar, escalar en cuatro verticales.

Caso real →

Chapter 05

Hoja de ruta de adopción

La hoja de ruta de cinco fases a lo largo de doce meses, de Descubrir a Operar, criterios de salida explícitos por fase, y coste medido por loop en GitHub AI Credits.

Hoja de ruta →

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 ANATOMÍA DEL AGENTE

Qué es un agente de IA, y qué no es: percibir, decidir, actuar, delimitado por política.

Un agente de IA percibe, decide y actúa, y su autonomía está delimitada por política. Esa única frase separa a los agentes de hoy de los chatbots de ayer, y prepara todo lo que sigue. Este capítulo da la definición de trabajo, contrasta un chatbot sin estado con un agente orientado a metas, nombra los cinco componentes que todo agente comparte y los ubica en un stack de decisión de cuatro capas, todo visto a través de GitHub Copilot.

Percibir, decidir, actuar: la definición de trabajo, delimitada por política

Un agente de IA es un software que percibe, decide y actúa. Su autonomía está delimitada por política. Esa definición es corta a propósito, y cada palabra pesa. Percibir significa que el agente lee su entrada: un prompt, un archivo, un repositorio, un evento, el resultado de su última acción. Decidir significa que planifica el siguiente paso hacia una meta, en lugar de devolver una única respuesta prefabricada. Actuar significa que cambia algo en el mundo: edita un archivo, llama a una API, ejecuta un comando, abre un pull request. La línea entre lo que existía antes y lo que las empresas despliegan hoy pasa exactamente por esos tres verbos.

La cuarta cláusula es la que los equipos olvidan. La autonomía está delimitada por política, y el límite es parte de la definición, no un control que agregas después. GitHub Copilot en agent mode lo hace concreto. Lee tu repositorio, planifica un cambio, edita archivos, ejecuta el build y abre un pull request, y hace todo eso dentro de los límites que escribes en .github/copilot-instructions.md. Ese archivo de instrucciones es donde vive la política: qué puede tocar el agente, qué convenciones debe seguir, cuándo debe detenerse y preguntar. Un agente sin política no es más capaz. Solo es un agente sin supervisión.

> DEFINICIÓN
>
> Un agente de IA percibe, decide y actúa, con autonomía delimitada por política. La plataforma que eliges define el límite. La arquitectura que entregas gana la confianza.

Chatbot versus agente: sin estado y reactivo versus orientado a metas, con estado y usando herramientas

El chatbot de ayer era sin estado y reactivo. Recibía una pregunta y devolvía una única respuesta. No guardaba memoria entre turnos, no podía llamar herramientas y no tenía noción de perseguir una meta. Pregúntale dos veces y empieza de cero cada vez. Ese diseño servía para preguntas frecuentes y flujos de soporte guionizados, y sigue sirviendo para eso. No es un agente, y llamarlo así crea la expectativa equivocada.

El agente de hoy está orientado a metas, mantiene estado y usa herramientas. Recibe un objetivo, planifica un camino para alcanzarlo, llama herramientas externas, observa los resultados e itera hasta cumplir la meta, o escala a un humano cuando no puede continuar de forma segura. La diferencia no es el modelo. El mismo modelo en el Copilot picker puede impulsar una respuesta de una sola vez o un agente de múltiples pasos. La diferencia es el loop a su alrededor y el estado que mantiene entre pasos. Cuando GitHub Copilot pasa de una respuesta de chat al agent mode, el modelo no cambia; el loop, la memoria y el acceso a herramientas sí.

Los cinco componentes que todo agente comparte: Brain, Memory, Tools, Policy, Orchestration

Todo agente, sin importar la plataforma, se construye a partir de cinco componentes. El Brain es el modelo que razona sobre la entrada y planifica el siguiente paso, elegido según la tarea. En GitHub Copilot el Brain es el modelo que seleccionas en el

Copilot picker: Claude Opus 4.8 para razonamiento profundo, Claude Fable 5 o Sonnet 5 para trabajo equilibrado, Haiku 4.5 para turnos rápidos y baratos, Gemini 3.1 Pro, o un modelo GPT-5.x. Eliges según la tarea, no una vez para todo el programa. La Memory es el historial de conversación, un bloc de notas de trabajo, un vector store para recuperación y memoria episódica entre ejecuciones.
La Memory está delimitada por política: un agente recuerda solo lo que tiene permitido recordar.

Tools, Policy y el loop que lo ata todo

Las Tools son las funciones, APIs, consultas a bases de datos, ejecución de código y activadores de workflow que el agente puede llamar. Están catalogadas y con alcance definido mediante el Model Context Protocol: cada servidor MCP es un conjunto versionado y nombrado de herramientas con un límite definido, para que el agente nunca hable con un endpoint aleatorio. La Policy es RBAC, filtros de contenido, límites de velocidad, puntos de control Human-in-the-Loop y registros de auditoría, y es no negociable; en GitHub Copilot el archivo .github/copilot-instructions.md y las reglas de tu organización la sostienen, y Azure AI Content Safety aplica los filtros de contenido. La Orchestration es el loop plan-act-observe, la coordinación multi-agente, los reintentos y los timeouts, y es responsabilidad de la plataforma. El Brain razona. La Memory persiste. Las Tools actúan sobre el mundo. La Policy delimita. La Orchestration impulsa el loop.

El stack de decisión como modelo mental: anatomía, luego plataforma, luego tipo, luego Human-in-the-Loop

Sostén esas partes en un stack de decisión de cuatro capas, porque cada capa restringe a la de abajo. La Capa I es la Anatomía: percibir, decidir, actuar, y los cinco componentes. La Capa II es la Plataforma: Copilot Studio para citizen developers, GitHub Copilot Agents para equipos de ingeniería, Azure AI Foundry para producción regulada. La Capa III es el Tipo: desarrollo, productividad, proceso de negocio, o datos y analítica, emparejado con la familia de tarea y la persona dueña. La Capa IV es Human-in-the-Loop: cuatro cuadrantes definidos por impacto y reversibilidad, de la autonomía total a la aprobación por dos personas.

Lee el stack de abajo hacia arriba, y construye en ese orden. Acierta la anatomía primero, porque una elección de plataforma no arregla una definición rota. Luego elige la plataforma, luego empareja el tipo de agente con el trabajo, luego define el límite de Human-in-the-Loop que exige el riesgo de la acción. Salta una capa y el agente llega a producción sin supervisión. La presión por saltar es real: Gartner proyecta que el 40% de las aplicaciones empresariales tendrán agentes de IA de tarea específica para 2026, frente a menos del 5% en 2025. La anatomía es la capa que no puedes saltar. Todo lo demás en este playbook se apoya en ella.

Referencias

Fuentes primarias para la definición de trabajo, los cinco componentes y las referencias de plataforma y protocolo usadas a lo largo de este capítulo.

- GitHub Copilot Documentation, la plataforma de referencia para agent mode, agentes personalizados y el archivo copilot-instructions usado a lo largo de este playbook.
- Model Context Protocol Specification, el estándar abierto que cataloga y define el alcance de las herramientas que un agente puede llamar.
- Microsoft Copilot Studio Documentation, la plataforma low-code para usuarios de negocio y citizen developers.
- Azure AI Foundry Documentation, la plataforma para orquestación multi-agente y agentes de nivel producción.
- What is Azure AI Content Safety?, los filtros de contenido que aplican parte de la capa de política de un agente.
- Gartner, task-specific AI agents in 40% of enterprise apps by 2026, la proyección de adopción: el 40% de las aplicaciones empresariales tendrán agentes de IA de tarea específica para 2026, frente a menos del 5% en 2025.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

01 ELECCIÓN DE PLATAFORMA

Tres plataformas, un framework de decisión: Copilot Studio, GitHub Copilot y Azure AI Foundry.

Microsoft entrega tres plataformas para construir agentes de IA: Copilot Studio, GitHub Copilot y Azure AI Foundry. Son complementarias, no competidoras, y la mayoría de las empresas usa las tres. La pregunta difícil nunca es qué plataforma gana. Es qué caso de uso va a cada plataforma. Este capítulo entrega un framework de decisión construido sobre cuatro ejes: público, nivel de habilidad, tiempo hasta el piloto y modelo de gobernanza.

Tres plataformas, tres públicos

Copilot Studio está orientado a usuarios de negocio y desarrolladores ciudadanos. Ofrece autoría de temas mediante drag-and-drop, conectores nativos para Microsoft 365, Dataverse y Power Platform, y un ciclo de despliegue medido en horas a días. Alguien que nunca abrió un IDE puede publicar un agente funcional, y ese es el punto. Copilot Studio encaja cuando quien construye está más cerca del proceso de negocio que de la base de código, y cuando el agente vive dentro de Microsoft 365 y no dentro de un repositorio. El autoservicio de RRHH, la automatización del helpdesk de TI y los bots de atención al cliente son su hogar natural.

GitHub Copilot está orientado a ingenieros de software que trabajan dentro del IDE y el repositorio. El agent mode lee el código, planifica un cambio, edita archivos, ejecuta comandos y abre un pull request, todo dentro del flujo del desarrollador. Azure AI Foundry está orientado a ingenieros de ML y equipos de plataforma. Ofrece el SDK completo, un catálogo de modelos, orquestación multi-agente, pipelines de fine-tuning y observabilidad de nivel producción. El tiempo hasta el piloto en Foundry corre en semanas, pero el techo es mucho más alto. Las tres plataformas son complementarias. Copilot Studio es dueño de la superficie conversacional,

GitHub Copilot es dueño del ciclo de ingeniería, y Azure AI Foundry es dueño de los agentes verticales que cargan SLAs estrictos.

Elegir por público, habilidad, tiempo hasta el piloto y gobernanza

Cuatro ejes deciden a dónde va un caso de uso. El público va primero: negocio y desarrolladores ciudadanos apuntan a Copilot Studio, ingenieros de software apuntan a GitHub Copilot, y equipos de ML o plataforma apuntan a Azure AI Foundry. El nivel de habilidad va segundo: Copilot Studio es low-code y no-code, GitHub Copilot es code-first y nativo en el IDE, y Azure AI Foundry es pro-code con SDK completo. Lee esos dos ejes juntos y la mayoría de las decisiones se resuelven solas. Quién construye y dónde se construye ya dicen qué plataforma abrir.

El tiempo hasta el piloto es el tercer eje. Copilot Studio entrega en horas a días, GitHub Copilot en días, y Azure AI Foundry en semanas. La gobernanza es el cuarto, y los equipos la subestiman. Copilot Studio hereda la administración de Power Platform, GitHub Copilot hereda las políticas de GitHub, y Azure AI Foundry hereda el Azure RBAC y el Content Safety. Elige la plataforma cuya gobernanza nativa ya corresponde al riesgo del caso de uso. Adaptar controles después del piloto cuesta más que elegir bien el primer día. Gartner prevé que el 40% de las aplicaciones empresariales tendrán agentes de IA específicos por tarea para 2026, frente a menos del 5% en 2025, así que el volumen de estas decisiones está por crecer.

El agent mode de GitHub Copilot en el repositorio

El agent mode es donde GitHub Copilot deja de sugerir líneas y empieza a hacer el trabajo. Lee el repositorio, planifica un cambio, edita archivos, ejecuta comandos y abre un pull request para revisión. El comportamiento lo moldea un archivo: .github/copilot-instructions.md. Ese archivo carga las convenciones que la base de código ya codifica, manejo de excepciones, formato de logging, organización de pruebas y nomenclatura, para que cada sesión de agent mode las herede en lugar de adivinar. Escríbelo una vez y cada tarea empieza fundamentada. Las herramientas MCP extienden el agente más allá del repositorio. Cada servidor del Model Context Protocol está catalogado, versionado y con alcance definido, dando

al agente acceso a sistemas de build, rastreadores de tickets y servicios internos sin hablar nunca con una API aleatoria.

El selector de modelos de Copilot convierte la elección de modelo en una decisión por tarea. Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y GPT-5.x están todos en el mismo selector, y diriges la tarea al modelo que combina con su costo y su dificultad. Desde el 1 de junio de 2026, el uso de modelos premium se factura como GitHub AI Credits, donde 1 crédito equivale a US$ 0,01. Cada sesión de agent mode y cada tarea en segundo plano del Coding Agent es una línea de factura medida. Trata esos créditos como gasto de infraestructura, presupuéstalos por equipo, y la historia de FinOps se mantiene honesta.

> DEFINICIÓN
>
> El agent mode se fundamenta en .github/copilot-instructions.md y alcanza el mundo externo por servidores MCP con alcance definido. La elección de modelo es una decisión por tarea en el selector de Copilot, y cada llamada premium se mide como GitHub AI Credits a 1 crédito = US$ 0,01.

Dónde aterriza cada caso de uso

Mapea los cuatro ejes sobre el trabajo real y las fronteras se vuelven nítidas. El autoservicio de RRHH y la automatización del helpdesk de TI aterrizan en Copilot Studio: quien construye es un dueño de negocio, los conectores a Microsoft 365 son nativos, y el piloto entrega en horas a días. La generación de código, la refactorización, la automatización de PRs y la generación de pruebas aterrizan en el agent mode de GitHub Copilot: el trabajo vive en el repositorio, las convenciones viven en .github/copilot-instructions.md, y el pull request es el checkpoint. Ninguno de estos casos de uso necesita un SDK completo, y forzarlos hacia uno desperdicia semanas.

Los agentes verticales regulados aterrizan en Azure AI Foundry. Un flujo de KYC y AML en finanzas, un flujo de autorización previa en salud, o un flujo fiscal de NF-e con un plazo legal de 15 días exigen orquestación multi-agente, Azure RBAC, Content Safety y Human-in-the-Loop con control dual en las acciones irreversibles.

Ese es el techo que solo Foundry ofrece, y las semanas de tiempo hasta el piloto compran los SLAs que la vertical exige. La regla que vale para las tres plataformas es la misma: empieza estrecho. Un caso de uso, un agente, un equipo, y generaliza solo después de que el valor esté demostrado.

Un parque, un framework

La mayoría de las empresas no elegirá una sola plataforma. Correrá Copilot Studio, GitHub Copilot y Azure AI Foundry lado a lado, cada uno cargando los casos de uso que le combinan. El framework es lo que mantiene ese parque coherente: los mismos cuatro ejes, aplicados por caso de uso, para que cada agente aterrice en la plataforma cuyo público, nivel de habilidad, tiempo hasta el piloto y gobernanza ya le corresponden.

Referencias

Fuentes primarias para las tres plataformas, su modelo de gobernanza y los patrones de integración de agentes de este capítulo.

- Microsoft Copilot Studio documentation, la plataforma low-code para usuarios de negocio y desarrolladores ciudadanos.
- GitHub Copilot documentation, la referencia para el agent mode, el .github/copilot-instructions.md, las herramientas MCP y el selector de modelos.
- Azure AI Foundry documentation, la plataforma pro-code para orquestación multi-agente y agentes verticales regulados.
- What is Azure AI Content Safety?, el control de gobernanza que Azure AI Foundry hereda para cargas reguladas.
- Model Context Protocol, el estándar abierto que da alcance y cataloga las herramientas que un agente puede llamar.
- Gartner, agentes de IA específicos por tarea en el 40% de las aplicaciones empresariales para 2026, la proyección de mercado detrás del volumen de decisiones de plataforma que viene.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

02 MÉTODO Y ARQUITECTURA

Tipos de agente y la arquitectura de seis capas: ajusta el trabajo, luego apila la plataforma.

Un programa de agentes enterprise tiene dos decisiones de diseño que acertar antes de que cualquier código llegue a producción. La primera es el tipo de agente, que se deriva del trabajo y de la persona que es dueña de él. La segunda es la arquitectura, seis capas apiladas desde la gobernanza hasta la superficie donde el usuario se encuentra con el agente. Este capítulo define los cuatro tipos, recorre las seis capas de abajo hacia arriba y muestra dónde encajan herramientas, modelos y contexto.

Cuatro tipos de agente, ajustados al trabajo

Cuatro tipos de agente cubren la mayor parte de la demanda enterprise, y cada uno se mapea claramente en una plataforma. Los agentes de desarrollo viven en el repositorio y el IDE. GitHub Copilot en agent mode es el entorno natural, con instrucciones de repositorio en .github/copilot-instructions.md y herramientas MCP conectadas a sistemas de build y gestores de tickets. Las tareas típicas son corrección de errores, refactorización y generación de pruebas. Los agentes de productividad operan dentro de Microsoft 365. Copilot Studio gestiona la autoría de temas personalizados, y Microsoft 365 Copilot Agents entrega experiencias embebidas en Teams, Outlook y SharePoint. Las tareas típicas son notas de reuniones, borradores de correo y búsqueda en documentos.

Los otros dos tipos cargan más riesgo. Los agentes de procesos de negocio orquestan entre sistemas. Copilot Studio es dueño de la capa conversacional, y Azure AI Foundry toma el relevo cuando el flujo exige SLAs estrictos, coordinación multi-agente o fine-tuning. Las tareas típicas son incorporación de empleados, flujos de aprobación y procesamiento de reclamaciones. Los agentes de datos y analítica fundamentan sus respuestas en datos empresariales alojados en Fabric, Synapse o

un almacén de datos propio. Exigen ingeniería de prompt rigurosa, un harness de evaluación sólido y aprobación humana antes de publicar cualquier salida. Las tareas típicas son lenguaje natural a SQL y resúmenes de insights. El tipo no es una etiqueta. Fija la gobernanza que le debes al agente.

Las seis capas, de abajo hacia arriba

Los programas de agentes de producción se construyen en seis capas, y cada capa restringe a la que está encima. La base es Plataforma y Gobernanza: identidad, RBAC, Content Safety, auditoría, observabilidad y FinOps. Son los mismos controles que ya operan el resto de Azure, no un stack paralelo. Encima están los Modelos: un catálogo de modelos elegidos por tarea, con fine-tuning solo donde el retorno justifica el costo. Encima de los Modelos viene Contexto y RAG: un vector store, recuperación semántica y acceso con alcance por identidad a los datos empresariales. Estas tres capas deciden lo que el agente sabe antes de actuar.

Herramientas, orquestación y la superficie

Las tres capas de arriba deciden lo que el agente hace. Herramientas y MCP es el registro de servidores MCP y plugins catalogados, versionados y con alcance definido, que permiten al agente actuar sobre el mundo. Orquestación es el runtime: el ciclo planificar, actuar, observar, coordinación multi-agente, reintentos, puntos de control HITL y evaluación. Superficie es donde los usuarios se encuentran con el agente, en Teams, Outlook, el IDE o una app personalizada. Salta una capa y la brecha no desaparece. Se va a producción, sin supervisión. Empieza estrecho, prueba un caso de uso en las seis capas, luego generaliza.

> DEFINICIÓN
>
> Seis capas, de abajo hacia arriba: Plataforma y Gobernanza, Modelos, Contexto y RAG, Herramientas y MCP, Orquestación y Superficie. Cada capa restringe a la que está encima. Salta una y el agente llega a producción sin el control que esa capa debía imponer.

Catalogar herramientas mediante MCP

Las herramientas son donde el agente deja de conversar y empieza a actuar, así que esta es la capa que necesita la disciplina más estricta. Cada herramienta que un agente puede llamar es una función, una API, una consulta a base de datos o un disparador de workflow, y cada una se expone mediante el Model Context Protocol. MCP es el estándar abierto de cómo un agente descubre y llama herramientas, y convierte un conjunto ad hoc de integraciones en un catálogo gobernado. La regla es simple. Ningún agente habla con una API aleatoria. Cada herramienta se registra en un registro de servidores MCP, se versiona y se acota al permiso más pequeño que resuelve la tarea.

El catálogo carga tres obligaciones. Registrar cada servidor para que un equipo de plataforma vea lo que existe y retire lo que queda obsoleto. Acotar cada token para que el agente tenga solo el acceso que su tarea exige, nunca los derechos completos del usuario. Auditar cada llamada para que un revisor pueda reconstruir lo que el agente hizo y por qué. En GitHub Copilot, los mismos servidores MCP sostienen el agent mode en el IDE, lo que significa que un agente de desarrollo y un agente de procesos de negocio toman herramientas de una única fuente gobernada, y no de dos divergentes. Una herramienta catalogada es revisable. Una no catalogada es un pasivo que espera un incidente.

Elegir el modelo por tarea en el selector de Copilot

El modelo es una elección, no un valor por defecto. En GitHub Copilot el selector expone varios modelos, y el correcto depende de la tarea que tienes delante. Claude Fable 5, Opus 4.8, Sonnet 5 y Haiku 4.5 conviven con Gemini 3.1 Pro y la familia GPT-5.x. Una refactorización corta con un presupuesto de latencia ajustado no necesita el mayor modelo de razonamiento. Una migración multi-archivo con dependencias profundas sí. Ajusta el modelo a la dificultad de la tarea, y vuelve a revisar el ajuste cuando cambie la tarea o el modelo.

El costo es parte de la elección. GitHub Copilot factura el uso de modelos en GitHub AI Credits, donde un crédito equivale a un centavo de dólar, en vigor desde el 1 de junio de 2026. Un modelo más grande gasta más créditos por solicitud, así que el

selector también es una decisión de presupuesto, no solo de capacidad. Trata cada cambio de modelo como una modificación que necesita prueba. Mantén un golden test set por agente, ejecuta regresión en cada cambio y sostén un presupuesto de latencia por ciclo. Un modelo que pasa el eval gana el tráfico. Uno que no pasa se queda fuera, sea cual sea su reputación.

Contexto y RAG del modo correcto

La capa de Contexto y RAG decide lo que el agente sabe antes de razonar. Bien hecha, fundamenta las respuestas en datos empresariales y corta las alucinaciones que surgen cuando un modelo adivina hechos que nunca recibió. Tres piezas la hacen funcionar. Un vector store guarda embeddings del corpus empresarial, desde documentos hasta código y tickets. La recuperación semántica trae los pasajes que de verdad importan para la solicitud, no los que solo comparten palabras clave. Y el acceso con alcance por identidad garantiza que el agente recupere solo lo que el usuario que lo invoca tiene permiso de ver.

Esa tercera pieza es la que los equipos saltan, y es la que más importa en un parque regulado. La recuperación con alcance por identidad significa que el vector store honra el mismo RBAC que gobierna los sistemas de origen. Un agente que actúa por un ingeniero de soporte no puede sacar a la luz un documento que el ingeniero jamás podría abrir directamente. Conecta la recuperación a la capa de identidad de la base de Plataforma y Gobernanza, no por fuera de ella. Un agente de datos y analítica que se fundamenta en Fabric o Synapse todavía debe aprobación humana antes de publicar, porque fundamentar baja la tasa de alucinación pero no la lleva a cero.

Referencias

Fuentes primarias para las plataformas, el protocolo de herramientas y los controles de gobernanza que usa este capítulo.

- Microsoft Copilot Studio documentation, la plataforma para agentes de productividad y de procesos de negocio construidos por citizen developers.

- GitHub Copilot documentation, agent mode, instrucciones de repositorio y el selector de modelos para agentes de desarrollo.
- Azure AI Foundry documentation, el catálogo de modelos y el runtime multi-agente para agentes verticales con SLAs estrictos.
- What is Azure AI Content Safety?, un control en la capa de Plataforma y Gobernanza en la base de la arquitectura.
- Model Context Protocol, el estándar abierto de cómo los agentes descubren y llaman herramientas catalogadas.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, la señal de demanda: agentes de IA task-specific proyectados para alcanzar el 40% de las apps enterprise para 2026.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps

03 GUARDRAILS Y GOBERNANZA

Human-in-the-Loop y gobernanza reutilizada: guardrails que escalan.

Más autonomía no compra menos controles. Exige más. Un agente que puede abrir un pull request, mover dinero o modificar un registro de cliente necesita puntos de control humanos explícitos, ubicados por impacto y reversibilidad y no por intuición. Este capítulo mapea los cuatro cuadrantes de Human-in-the-Loop, los tres patrones de implementación que cubren la mayoría de los casos, y la gobernanza que la empresa ya tiene en Azure y reutiliza en lugar de reconstruir.

Más autonomía exige más guardrails: los cuatro cuadrantes

Más autonomía requiere más guardrails, no menos. El coste de una acción autónoma incorrecta supera el ahorro de cien acciones correctas, así que un agente que puede actuar sobre datos regulados, firmar documentos, mover dinero o modificar registros de clientes necesita puntos de control humanos explícitos. Dónde ubicas esos puntos de control no es cuestión de gusto. Se deriva de dos propiedades de cada acción: impacto de negocio y reversibilidad. El impacto es lo que se rompe si la acción es incorrecta. La reversibilidad es qué tan barato es deshacerla. Cruza esos dos ejes y cada acción cae en uno de cuatro cuadrantes, cada uno con su propio modo de revisión.

El cuadrante uno, Automatizar, reúne acciones de bajo impacto y alta reversibilidad: sugerir un mensaje de commit, redactar una respuesta de correo, resumir una reunión, buscar documentos. El agente ejecuta estas acciones dentro de su propio ciclo. Auditas el log, pero no añades punto de control. El cuadrante dos, Revisar, reúne acciones de alto impacto y alta reversibilidad: abrir un pull request, programar una reunión, enviar un borrador a un cliente. Un humano aprueba antes del commit o del envío. El cuadrante tres, Confirmar, reúne acciones de bajo impacto y baja reversibilidad: eliminar un archivo temporal, archivar un ticket, cerrar una

conversación. Un único paso de confirmación basta. El cuadrante cuatro, Control Dual, reúne acciones de alto impacto y baja reversibilidad: firmar un contrato, transferir fondos, modificar RBAC, eliminar datos de producción, presentar impuestos. Dos humanos aprueban, y el rastro de auditoría lleva evidencia firmada. La mayoría de los agentes corporativos pertenece al cuadrante dos o tres, no al uno.

> DEFINICIÓN
>
> Los cuatro cuadrantes en una línea: Automatizar para bajo impacto y reversible, Revisar para alto impacto y reversible, Confirmar para bajo impacto e irreversible, Control Dual para alto impacto e irreversible. Ubica el punto de control por impacto y reversibilidad, nunca por intuición.

Tres patrones de implementación: aprobar antes, muestrear y revisar, y control dual

Tres patrones de implementación cubren la mayor parte de lo que exigen los cuadrantes. Aprobar antes hace que el agente proponga una acción y espere en un punto de control antes de actuar. Sirve para pull requests, borradores y recomendaciones, y se mapea directamente en el agent mode de GitHub Copilot: el agente planifica y edita, luego se detiene para que un humano acepte o rechace antes de cualquier merge. Muestrear y revisar deja que el agente actúe en alto volumen mientras una muestra aleatoria de su salida se audita y un umbral de KPI dispara la escalada. Sirve para etiquetado, clasificación y enrutamiento. Control dual requiere dos personas distintas, un iniciador y un aprobador separado, con una firma criptográfica de cada lado. Sirve para finanzas, legal y datos de producción, donde el registro de quién aprobó qué tiene que sobrevivir a una auditoría.

Los patrones también muestran cómo escalar la revisión sin convertirse en el cuello de botella. Aprobar antes se mantiene barato cuando la propuesta del agente es legible: un diff claro, una justificación corta y un enlace a la evidencia usada. Codifica esa expectativa una vez en .github/copilot-instructions.md para que cada ejecución produzca el mismo formato revisable. Muestrear y revisar mantiene rápidos el cuadrante uno y las partes más ligeras del cuadrante tres: aceptas que algunas acciones de bajo impacto se envían sin un humano, y defiendes esa

elección con el log de auditoría y la tasa de muestreo. Control dual es innegociable para el cuadrante cuatro, y es donde la evidencia firmada justifica su coste. El coste es, en sí mismo, un control. GitHub Copilot factura el trabajo de agente en GitHub AI Credits, donde un crédito equivale a un centavo de dólar, así que un presupuesto de créditos por ejecución es un guardrail cuantitativo que defines y auditas junto a los puntos de control humanos.

Reutiliza la gobernanza de Azure en lugar de crear controles paralelos

La forma más rápida de volver inseguro un programa de agentes es construir un segundo conjunto de controles junto a los que la empresa ya confía. Los controles que un agente necesita son los controles que Azure ya ofrece. La identidad es Entra ID, el mismo directorio que gobierna cualquier otro workload. La autorización es RBAC, con alcance en lo que la tarea exige y nada más. Content Safety filtra prompts y respuestas contra las categorías que la organización ya definió. Purview clasifica y rastrea los datos que el agente toca, para que un registro regulado siga regulado cuando un agente lo lee. La auditoría es por llamada: cada invocación de herramienta, cada solicitud de modelo, cada aprobación humana cae en el mismo log que escribe el resto del parque. Nada de esto es específico de agentes. Es la gobernanza permanente de la plataforma, apuntada a un nuevo workload.

El acceso a herramientas es la parte que la mayoría de los equipos subestima. Un agente es tan seguro como las herramientas que puede llamar, así que cada herramienta se cataloga, se versiona y se le define alcance mediante el Model Context Protocol antes de que un agente pueda alcanzarla. Ningún agente habla con una API aleatoria. En GitHub Copilot, los servidores MCP que un agente puede usar se declaran en configuración y se restringen con tokens de alcance, de modo que el radio de impacto de cualquier herramienta se conoce de antemano. Los modelos se gobiernan igual: los que un agente puede seleccionar vienen del selector de GitHub Copilot, un conjunto fijo que hoy incluye Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y la familia GPT-5.x, cada uno con un coste conocido en GitHub AI Credits. Reutilizar identidad, autorización, content safety, gobernanza de datos, auditoría y un catálogo de herramientas con alcance hace que el agente herede la postura de cumplimiento que la empresa ya opera.

Cinco principios para un programa de agentes gobernado

Cinco principios sostienen un programa de agentes de Microsoft, y cada uno se deriva de las dos secciones anteriores. Empieza estrecho: un caso de uso, un agente, un equipo, y generaliza solo después de demostrar el valor, porque un alcance estrecho es un radio de impacto pequeño. Reutiliza la gobernanza: identidad, RBAC, Content Safety y Purview son los mismos controles del resto del stack, no paralelos, y cada control paralelo que evitas construir es uno que nunca tienes que auditar dos veces. Cataloga cada herramienta: un registro de servidores MCP, tokens con alcance y auditoría por llamada garantizan que ningún agente alcance una capacidad que nadie aprobó. Estos tres principios tratan de contención. Mantienen el conjunto de cosas que un agente puede hacer pequeño, conocido y revisable.

Los dos últimos principios tratan del tiempo. Evalúa antes de desplegar: mantén un golden test set por agente, ejecuta regresión en cada cambio de modelo y respeta un presupuesto de latencia por ciclo, para que un cambio en el modelo o en el prompt no altere en silencio un comportamiento que un revisor ya aprobó. Cambiar de un modelo del selector de Copilot a otro es un cambio de modelo, y pasa por la misma puerta de evaluación que cualquier otro cambio. Planifica la retirada: versiona cada prompt, registra cada cambio y mantén el rollback, porque los agentes derivan igual que los modelos, y un programa sin plan de salida acumula riesgo que no puede ver. Los guardrails no son una lista de verificación de lanzamiento que cumples una vez. Son la forma permanente del programa, actualizada a medida que los agentes cambian.

Referencias

Fuentes primarias para la plataforma, los guardrails y los controles de gobernanza que este capítulo reutiliza.

- GitHub Copilot Documentation, agent mode, archivos de instrucciones del repositorio y el selector de modelos que este capítulo gobierna.
- Azure AI Foundry Documentation, gobernanza de producción para agentes corporativos: identidad, RBAC y observabilidad.

- What is Azure AI Content Safety?, la capa de filtrado de contenido reutilizada como guardrail de agente, no reconstruida.
- Model Context Protocol, el estándar abierto para catalogar y dar alcance a las herramientas que un agente puede llamar.

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 PRUEBA DE EJECUCIÓN

Arrendamiento de vehículos en producción, y más allá.

Dos problemas operativos en arrendamiento de vehículos muestran cómo se ve un diseño multi-agente una vez que sale de la diapositiva y corre en producción. Uno es un plazo de multa medido en días. El otro es un documento fiscal que bloquea un mes de facturas cuando un código está mal. Los dos se resuelven en los mismos cuatro movimientos, y los mismos cuatro movimientos sirven para otras cuatro verticales. Este capítulo es la prueba de ejecución.

Indicación de conductor en multas: cuatro agentes contra un plazo de 15 días

El primer caso corrió en arrendamiento de vehículos, y empezó con un reloj legal. Cuando un coche arrendado recibe una multa de tráfico, la ley da a la arrendadora 15 días para indicar al conductor que iba al volante. Pierde esa ventana y la empresa absorbe la multa más una penalización encima. El proceso manual era frágil. El área de correspondencia recibía el aviso en papel, un operador lo escaneaba y retecleaba los campos en una hoja de cálculo, luego enviaba un correo al cliente que arrendó el coche y esperaba respuesta. Los plazos se incumplían con frecuencia suficiente para volverse una partida de costo. Cuatro agentes especializados reemplazaron ese flujo, y corren en Azure AI Foundry.

Cada agente es dueño de una tarea. Un agente de ingestión usa Document Intelligence para extraer la matrícula, la fecha de la infracción, el código, el valor y el plazo del aviso escaneado. Un agente de consulta cruza SAP y Salesforce para encontrar al arrendatario activo. Un agente de notificación envía enlace al portal, SMS y correo. Un agente de seguimiento vigila el reloj: escala el día 7, envía una alerta final el día 12 y deriva el caso a un paralegal el día 14. La plantilla de notificación queda detrás de un checkpoint Human-in-the-Loop, así que el equipo de operaciones aprueba el texto antes de que cualquier mensaje llegue al cliente.

Nada aquí es exótico. El valor viene de dividir una cadena manual frágil en cuatro agentes que hacen, cada uno, una cosa estrecha y comprobable.

Un detalle de implementación importa para el costo y la repetibilidad. Los cuatro agentes se especificaron y construyeron con GitHub Copilot en agent mode. Cada herramienta que llaman se declara como un servidor MCP, las convenciones del arrendamiento viven en .github/copilot-instructions.md, y cada agente elige su modelo en el Copilot picker: Haiku 4.5 para extracción barata de campos, Sonnet 5 para la lógica de consulta y notificación, y Opus 4.8 donde el timing de escalada tiene que razonar sobre casos límite. El costo de build se mide en GitHub AI Credits, donde un crédito es un centavo. El mismo picker y el mismo archivo de instrucciones sirven para el flujo fiscal, y por eso el segundo caso se levantó en días, no en semanas.

Cumplimiento fiscal de NF-e: tres agentes bajo control dual

El segundo caso tiene un fallo más afilado. Cada cierre de contrato, devolución de vehículo y transferencia entre sucursales debe generar una Nota Fiscal Eletrônica válida, y las reglas vienen de la legislación del RICMS, que cambia por estado y por categoría de producto en la tabla NCM. Un código CFOP incorrecto puede bloquear facturas de todo un mes. La emisión manual aguanta en volumen bajo y se rompe a escala. Tres agentes llevan el flujo, activados por un evento de ERP de SAP, Oracle o TOTVS.

El primer agente es un clasificador fiscal. Usa Azure AI Search sobre el texto del RICMS y la tabla NCM para elegir el CFOP, el CST y el tipo de ICMS correctos, así que la clasificación queda anclada en la legislación real en vez de inferida. El segundo agente emite el documento contra la API SEFAZ NF-e 4.0 y gestiona reintentos, modo de contingencia y la ruta de cancelación. El tercer agente corre una reconciliación diaria sobre SPED, EFD-Contribuições, los libros de entradas y salidas y el informe de ICMS-ST. El cierre fiscal mensual no corre solo. Queda bajo un checkpoint de control dual: una persona inicia, una segunda aprueba, y el CFO firma con evidencia auditada. Alto impacto y baja reversibilidad merecen el guardrail más estricto del framework.

El patrón reutilizable: ingerir, clasificar, actuar, escalar

Mira los dos casos y aparece el mismo esqueleto. Un agente ingiere un documento o un evento. Un agente clasifica contra reglas que el negocio ya codifica, sea un registro de arrendatario o un código fiscal. Un agente actúa, enviando un aviso o emitiendo un documento fiscal. Un agente escala cuando un plazo se acerca o se exige una aprobación humana. Ingerir, clasificar, actuar, escalar. El patrón es deliberadamente aburrido, porque aburrido es lo que sobrevive a una auditoría y a una revisión de incidente en producción.

Lo que cambia entre casos no es la forma, es el guardrail. El flujo de multas revisa una plantilla y escala en un calendario fijo. El flujo de NF-e corre una reconciliación diaria y traba el cierre mensual detrás de control dual. Las dos decisiones salen directo del framework Human-in-the-Loop: mapea cada acción por impacto y reversibilidad, luego coloca el checkpoint donde una acción autónoma incorrecta costaría más que la revisión. El patrón te da los agentes. El framework te dice dónde un humano todavía tiene que estar.

> DEFINICIÓN
>
> Ingerir, clasificar, actuar, escalar. Cuatro agentes estrechos, un checkpoint Human-in-the-Loop colocado por impacto y reversibilidad. Empieza con un caso de uso y un equipo, prueba el valor en producción, luego reutiliza el esqueleto en la siguiente vertical.

Cuatro verticales sobre el mismo patrón

El patrón no se queda en arrendamiento de vehículos. En finanzas, un agente de documentos lee archivos de incorporación, un agente de puntuación de riesgo cruza nombres con listas de sanciones, y un agente de escalada dirige al oficial de cumplimiento, con control dual en la aprobación de KYC y AML. En salud, un agente de extracción clínica se une a una capa RAG de reglas de pagadores, y un médico revisa antes de cualquier envío de autorización previa. El tiempo de autorización cae

de días a horas, y cada decisión queda registrada para auditoría. Los mismos cuatro movimientos, reglas distintas, un checkpoint distinto.

Retail y manufactura lo repiten de nuevo. En retail, un agente de triaje clasifica el motivo de la devolución, un agente de reembolso corre la verificación de política, y un agente de logística programa la recogida, con el checkpoint Human-in-the-Loop reservado para excepciones por encima del umbral de valor del SKU. En manufactura, un agente de telemetría vigila señales de IoT, un agente de diagnóstico propone órdenes de trabajo, y un agente de programación asigna técnicos, con el gerente de planta aprobando cualquier orden mayor a cuatro horas. Cuatro verticales, un esqueleto: ingerir, clasificar, actuar, escalar, con el checkpoint donde el costo de equivocarse es más alto. Esa es la prueba. El patrón pasó de una demo a producción en arrendamiento, y se transfiere sin reescritura.

Referencias

Fuentes primarias para las plataformas, herramientas y controles de gobernanza detrás de los casos en producción.

- Azure AI Foundry documentation, el runtime donde corren los cuatro agentes de arrendamiento y los tres agentes de NF-e.
- GitHub Copilot documentation, el agent mode, el .github/copilot-instructions.md y el model picker usados para construir los agentes.
- Model Context Protocol, el estándar por el que cada herramienta de agente se declara y se acota.
- Azure AI Content Safety overview, los controles de contenido y seguridad que respaldan los checkpoints Human-in-the-Loop.
- Microsoft Copilot Studio documentation, la ruta low-code para la superficie conversacional de los agentes de proceso de negocio.

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 HOJA DE RUTA DE ADOPCIÓN

Del primer piloto a la flota en producción: cinco fases, criterios de salida explícitos.

Un piloto que funciona no es un programa en producción. La distancia entre un agente ayudando a un equipo y una flota gobernada que sirve a toda la empresa se mide en fases, no en funcionalidades. Este capítulo presenta la hoja de ruta de cinco fases que recorre doce meses, los criterios de salida que impiden que un piloto salga como una demo, y la disciplina de coste que mantiene una flota solvente cuando corre sobre GitHub AI Credits.

Las cinco fases: Descubrir, Piloto, Endurecer, Escalar, Operar

El camino desde el primer piloto hasta una flota de agentes de nivel producción recorre cinco fases a lo largo de doce meses. Cada fase cambia lo que construyes y lo que tienes que demostrar antes de avanzar. Descubrir ocurre en el mes uno. El Piloto, en los meses dos y tres. Endurecer, de los meses cuatro a seis. Escalar, de los meses siete a nueve. Operar comienza en el mes diez y no termina. La secuencia es deliberada, y no puedes saltarte una fase sin trasladar su coste a una fase posterior, donde siempre es mayor y más difícil de pagar.

Descubrir mapea los casos de uso candidatos por valor y esfuerzo, obtiene la aprobación de los stakeholders y selecciona un candidato al piloto. El Piloto despliega un único agente con un único equipo y ejecuta una evaluación en ciclo cerrado antes de que nadie más tenga acceso. Endurecer implementa RBAC y registros de auditoría, establece objetivos de SLA y topes de coste, y conecta puntos de control Human-in-the-Loop en cada acción que toca datos regulados o estado irreversible. Estas tres primeras fases son cómo una idea se convierte en algo que un comité de revisión acepta aprobar para usuarios reales.

Escalar construye el registro de herramientas, introduce orquestación multi-agente y abre incorporación self-service para que nuevos equipos traigan sus propios casos de uso sin un ingeniero de plataforma en el loop de cada uno. Operar establece las disciplinas que mantienen una flota sana durante años: FinOps, monitoreo de drift, simulacros de cambio de modelo y revisiones trimestrales contra los objetivos de valor que se escribieron allá en Descubrir. Escalar amplía la superficie del programa. Operar es lo que mantiene esa superficie honesta.

Criterios de salida explícitos por fase

Cada fase tiene una puerta de salida, y la puerta es un conjunto de condiciones, no una fecha en el calendario. Descubrir termina cuando se elige un caso de uso y su objetivo de valor queda escrito en un número por el que alguien responderá. El Piloto termina cuando la evaluación en ciclo cerrado supera su golden test set y el equipo patrocinador realmente quiere seguir usando el agente. Endurecer termina cuando toda acción de alto impacto y baja reversibilidad queda detrás de un punto de control de control dual y cada llamada que hace el agente se audita.

Los dos fallos que más veo son saltarse la puerta después del Piloto y saltarse la puerta después de Escalar. Pilotar sin endurecer produce una demo: funciona en la sala y se cae la primera vez que se topa con permisos reales, requisitos reales de auditoría y una factura real. Escalar sin operar produce deuda: más agentes, más equipos y más gasto, sin nadie vigilando el drift ni leyendo la cuenta del mes. Los criterios de salida existen para hacer visibles ambos fallos temprano, antes de que se acumulen por la flota.

> DEFINICIÓN
>
> Cada fase tiene criterios de salida explícitos. Pilotar sin endurecer produce una demo. Escalar sin operar produce deuda.

Operar para el largo plazo

Operar es donde la mayoría de los programas falla en silencio, porque el trabajo es poco vistoso y nunca acaba. Cuatro disciplinas lo sostienen. FinOps rastrea el gasto por agente, por equipo y por loop, y mantiene cada uno contra el tope de coste fijado en Endurecer. El monitoreo de drift vigila el deterioro lento que golpea a los agentes igual que golpea a los modelos: un prompt que funcionaba hace seis meses empieza a fallar, y el golden test set lo detecta antes que tus usuarios. Los simulacros de cambio de modelo ensayan cambiar el modelo por debajo de un agente sin cambiar cómo se comporta el agente.

Los simulacros de cambio de modelo importan porque el modelo detrás de un agente nunca queda fijo por mucho tiempo. En GitHub Copilot, el selector de modelos expone Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y la familia GPT-5.x. Una nueva entrada en el selector es una oportunidad de recortar coste o subir calidad, pero solo si puedes cambiarla y volver a ejecutar el golden test set para probar que el agente sigue comportándose. Ensaya ese cambio de forma programada, no durante un incidente. Las revisiones trimestrales entonces cierran el ciclo: mide cada agente contra su objetivo de valor original y retira los que ya no justifican su lugar.

Coste en la práctica: gasto por loop

Aquí el coste es una métrica de primera clase, no una sorpresa de fin de año. En GitHub Copilot, el uso de agentes se factura en GitHub AI Credits, y un crédito vale un centavo. Desde junio de 2026, esa es la unidad a la que convierte todo informe de FinOps. Rastrear el gasto por loop significa atribuir a cada ciclo de plan-act-observe un coste en créditos, para que un equipo vea qué agentes son baratos, cuáles son caros y cuáles son caros por una buena razón.

La disciplina se acumula a lo largo de toda la hoja de ruta. Endurecer fija el tope de coste. Escalar multiplica el número de loops que corren cada día. Operar lee el medidor y lo reporta. Un agente que cuesta un puñado de créditos por loop está bien en volumen de piloto y se convierte en una línea de coste real en volumen de flota, así que el presupuesto de latencia y de tokens que fijas por loop en Endurecer es el mismo presupuesto que mantiene la flota solvente en Operar. Coste por loop,

medido en créditos, es el número único que conecta el diseño de un agente con la cuenta del programa entero.

Referencias

Fuentes primarias para las plataformas, el modelo de facturación y los controles de gobernanza de los que depende esta hoja de ruta.

- GitHub Copilot Documentation, el agent mode y el modelo de facturación en GitHub AI Credits que mide el gasto por loop.
- Azure AI Foundry Documentation, el catálogo de modelos y la observabilidad de producción detrás de los simulacros de cambio de modelo y el monitoreo de drift.
- Microsoft Copilot Studio Documentation, el camino low-code para la incorporación self-service abierta en la fase de Escalar.
- Model Context Protocol, el estándar abierto detrás del registro de herramientas catalogado y con alcance construido en Escalar.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, la trayectoria de adopción enterprise que enmarca la hoja de ruta de doce meses.

Paula Silva

AI-NATIVE SOFTWARE ENGINEER

in/paulanunes

v1.0.0 · 2026-07-02

Building the future of software development with AI and Agentic DevOps