La plataforma es la primera decisión. Persona y radio de impacto deciden el resto.

Un agente de IA empresarial es un tipo distinto de software: mantiene un objetivo, guarda estado entre pasos, llama a herramientas para cambiar el mundo y se detiene ante un humano cuando no puede continuar de forma segura. Este playbook enruta cada carga de agente entre las tres plataformas de Microsoft, mapea los cuatro tipos de agentes que llegan a producción, presenta la arquitectura de referencia en seis capas, y muestra cómo la supervisión humana escala con la autonomía, 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

6
CAPÍTULOS

4
TIPOS DE AGENTES

40%

APPS EMPRESARIALES
PARA 2026

CAPÍTULOS

Chapter 00

Fundamentos de agentes

Qué separa a un agente de IA empresarial de un chatbot, los cinco componentes que todo agente reúne, y por qué la elección de plataforma va antes de la primera línea de código.

Fundamentos →

Chapter 01

Tres plataformas, un framework

Copilot Studio para usuarios de negocio, GitHub Copilot para ingenieros, Azure AI Foundry para equipos de ML y plataforma, y la matriz de seis ejes que enruta cada carga.

Plataformas →

Chapter 02

Cuatro tipos de agentes

Agentes de desarrollo, productividad, procesos de negocio y datos, la plataforma donde encaja cada uno, y el guardrail que cada tipo exige antes de ir a producción.

Tipos de agentes →

Chapter 03

La arquitectura de referencia

La pila de agentes enterprise en seis capas, cinco principios que rigen cada capa, tres patrones de orquestación y cuatro reglas de diseño que mantienen contenida una falla.

Arquitectura →

Chapter 04

Supervisión humana

Por qué más autonomía exige más guardrails, los cuatro cuadrantes de impacto y reversibilidad, tres patrones de punto de control, y la base de seguridad por debajo de ellos.

Supervisión →

Chapter 05

Del piloto a la flota

Un caso de leasing trabajado bajo gates Human-in-the-Loop, cuatro patrones verticales que reutilizan su coreografía, y una hoja de ruta de doce meses con criterios de salida.

Roadmap →

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 AGENT FOUNDATIONS

Qué es, en realidad, un agente de IA empresarial, y por qué la plataforma es la primera decisión.

Un agente de IA empresarial no es un chatbot más listo. Es un tipo distinto de software. Mantiene un objetivo, guarda estado entre pasos, llama a herramientas para cambiar el mundo y se detiene ante un humano cuando no puede continuar de forma segura. Este capítulo define qué es en realidad un agente, nombra los cinco componentes que todo agente reúne y muestra por qué la decisión de plataforma va antes de la primera línea de código, todo visto a través de GitHub Copilot.

Agente versus chatbot: orientado a objetivos, con estado y usando herramientas

Empieza por lo que un agente no es. Un chatbot es sin estado y reactivo. Recibe una pregunta y devuelve una única respuesta. No guarda memoria entre turnos, no puede llamar herramientas y no tiene noción de perseguir un objetivo. Pregúntale lo mismo dos veces y empieza de cero cada vez. Ese diseño sirve para preguntas frecuentes y flujos de soporte guionizados, y sigue sirviendo para eso. No es un agente. Un agente de IA empresarial está orientado a objetivos, mantiene estado y usa herramientas. Recibe un objetivo, planifica una secuencia de pasos, llama a herramientas, observa los resultados e itera hasta cumplir el objetivo. Cuando no puede continuar de forma segura, escala a un humano.

GitHub Copilot hace concreta la diferencia. El mismo modelo en el Copilot picker puede impulsar una respuesta de chat de una sola vez o un agente de múltiples pasos. Cuando Copilot pasa de una respuesta de chat al agent mode, el modelo no cambia; cambian el loop a su alrededor, el estado que guarda y las herramientas que puede llamar. En agent mode Copilot lee tu repositorio, planifica un cambio, edita archivos, ejecuta el build y abre un pull request. El escalado es parte del diseño, no

un agregado posterior: cuando el agente llega a una decisión que no puede tomar de forma segura, se detiene y pregunta. Un agente que nunca escala no es más autónomo. Solo es un agente sin supervisión.

Los cinco componentes que todo agente reúne: brain, memory, tools, policy y orchestration

Todo agente, sin importar la plataforma, se construye a partir de los mismos 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 conduce todo

Las tools son las funciones que el agente puede llamar: búsqueda, consultas a bases de datos, llamadas REST, ejecución de código, activadores de workflow. Están catalogadas y con alcance definido mediante el Model Context Protocol, para que el agente nunca hable con un endpoint aleatorio. La policy es RBAC, filtros de contenido, límites de tasa, checkpoints de Human-in-the-Loop y registros de auditoría, y es no negociable. En GitHub Copilot la policy vive en .github/copilot-instructions.md y en las reglas de tu organización. La orchestration es el loop plan-act-observe, la coordinación multi-agente, los reintentos y los timeouts, y es responsabilidad de la plataforma. El costo acompaña a todo esto: en GitHub Copilot, el trabajo del agente se factura en GitHub AI Credits, donde un credit equivale a un centavo de dólar, en vigor desde el 1 de junio de 2026, así que el brain que eliges y los tokens que gastas caen en la cuenta. El brain razona, la memory persiste, las tools actúan, la policy delimita, la orchestration conduce el loop.

Por qué la elección de plataforma es la primera decisión

La presión por poner agentes en producción es real, y se está acelerando. Gartner proyecta que el 40 por ciento de las aplicaciones empresariales tendrán agentes de IA de tarea específica para 2026, frente a menos del 5 por ciento en 2025. Es un salto grande en un solo año, y es donde la mayoría de los programas se equivoca. Los equipos parten de la herramienta que ya tienen abierta, en lugar de la decisión que aún no han tomado. La primera decisión no es qué modelo ni qué framework. Es en qué plataforma corre el agente, porque la plataforma define el límite de todo lo que el agente puede hacer.

El portafolio de agentes de Microsoft corre sobre tres plataformas, y son complementarias, no competidoras. Copilot Studio está dirigido a usuarios de negocio y citizen developers, con creación por arrastrar y soltar y conectores nativos con Microsoft 365. GitHub Copilot está dirigido a ingenieros de software, con agent mode, agentes personalizados y herramientas MCP que viven en el repositorio, junto al código que modifican. Azure AI Foundry está dirigido a ingenieros de ML y equipos de plataforma, con el SDK completo, el catálogo de modelos y orquestación multi-agente para agentes verticales de nivel productivo. La mayoría de las empresas termina corriendo las tres. La pregunta nunca es qué plataforma gana. Es qué caso de uso va a dónde.

La tesis del framework de decisión: la plataforma la define a quién sirve el agente y qué puede tocar

La tesis de este playbook cabe en una frase: la plataforma correcta la define a quién sirve el agente y qué puede tocar. A quién sirve es la persona. Un usuario de negocio creando un bot de helpdesk, un ingeniero refactorizando un servicio y un equipo de plataforma construyendo un agente de datos regulado son tres dueños distintos, con tres niveles técnicos distintos. Qué puede tocar es el radio de impacto. Un borrador de correo no es una transferencia de fondos, y un archivo temporal no es una base de datos de producción. Esos dos ejes, persona y radio de impacto, deciden más que cualquier comparación de funciones.

Lee los dos ejes juntos y la plataforma se revela. Un citizen developer que automatiza un flujo de aprobación dentro de Microsoft 365 pertenece a Copilot Studio. Un ingeniero que genera código y abre pull requests pertenece a GitHub Copilot, gobernado por .github/copilot-instructions.md y herramientas con alcance definido por MCP. Un equipo que construye un pipeline de datos multi-agente bajo SLAs estrictos pertenece a Azure AI Foundry, gobernado por Azure RBAC. Acierta la persona y el radio de impacto y todo lo que viene después tiene un hogar: el modelo en el Copilot picker, los checkpoints de Human-in-the-Loop, el registro de auditoría, la cuenta en GitHub AI Credits. Equivócalos y ninguna calidad de modelo rescata al agente. La plataforma es la primera decisión porque toda decisión posterior la hereda.

> DEFINICIÓN
>
> La tesis del framework de decisión en una frase: la plataforma correcta la define a quién sirve el agente y qué puede tocar. La persona define el nivel técnico y la superficie. El radio de impacto define los guardrails. Todo lo demás, el modelo, las herramientas, los checkpoints, el costo, se hereda de esos dos.

Referencias

Fuentes primarias para la definición de agente, el portafolio de plataformas y la proyección de adopción usada en 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.
- 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 productivo.
- Gartner, task-specific AI agents in 40 percent of enterprise apps by 2026, la proyección de adopción: el 40 por ciento de las aplicaciones empresariales

tendrán agentes de IA de tarea específica para 2026, frente a menos del 5 por ciento 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.

El portafolio de agentes de IA de Microsoft corre sobre tres plataformas, y cada una atiende a una audiencia diferente y a una superficie de control diferente. Copilot Studio es para usuarios de negocio y desarrolladores ciudadanos, GitHub Copilot es para ingenieros, y Azure AI Foundry es para ingenieros de ML y equipos de plataforma. Son complementarias, no competitivas, así que la decisión no es qué plataforma gana; es qué caso de uso va a dónde. 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, lo que convierte esa decisión de enrutamiento en una que todo equipo empresarial ahora debe tomar. Este capítulo pone las tres plataformas lado a lado y ofrece el framework para enrutar cada carga de trabajo.

Copilot Studio: para usuarios de negocio y desarrolladores ciudadanos

Copilot Studio es el punto de entrada para quienes son dueños de un proceso pero no escriben código. La creación es por arrastrar y soltar: construyes tópicos en un lienzo visual, los conectas a disparadores y los pruebas en la misma ventana. La biblioteca de conectores es la razón por la que encaja en entornos Microsoft. Los conectores nativos alcanzan Microsoft 365, Dataverse y Power Platform, así que un agente puede leer una lista de SharePoint, escribir en una tabla de Dataverse o iniciar un flujo de Power Automate sin ningún código de integración a medida. Quienes construyen estos agentes son las mismas personas dueñas del proceso de negocio, lo que elimina el traspaso a un equipo de ingeniería.

La gobernanza pasa por el centro de administración de Power Platform. Los administradores definen políticas de entorno, reglas de prevención de pérdida de datos y límites de uso compartido desde una sola consola, la misma consola que ya gobierna Power Apps y Power Automate en el tenant. El tiempo hasta un piloto

funcional se mide en horas o días. El intercambio es techo por velocidad: Copilot Studio es el hogar correcto para un asistente de RR. HH., un bot de helpdesk de TI o un flujo de atención al cliente en el que el equipo de negocio es dueño del agente, y es el hogar equivocado para un programa que necesita código de orquestación a medida o un modelo con fine-tuning.

GitHub Copilot: para ingenieros

GitHub Copilot es la plataforma para quienes escriben el código. El agent mode lleva a Copilot de sugerencias de una línea a trabajo de múltiples pasos: planifica un cambio, edita en varios archivos, corre la terminal, lee la salida e itera hasta que la tarea pasa. Lo diriges con un archivo del repositorio, .github/copilot-instructions.md, que declara las convenciones, los comandos de build y las restricciones que el agente debe respetar en cada tarea. Las herramientas del Model Context Protocol (MCP) extienden el agente más allá del repositorio, conectándolo a sistemas de build, rastreadores de issues y servicios internos por una única interfaz estándar, en lugar de una integración a medida por herramienta.

La elección de modelo está en el selector de Copilot. La lista actual incluye Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro y la familia GPT-5.x, y cambias de modelo por tarea sin salir del editor. La facturación corre sobre GitHub AI Credits, donde un crédito equivale a un centavo de dólar, en vigor desde el 1 de junio de 2026. Cada solicitud consume créditos según el modelo, así que un equipo presupuesta el uso del agente como presupuesta cualquier recurso medido, y un modelo más pesado cuesta más por tarea que uno más ligero. La gobernanza sigue el modelo de políticas de GitHub que la organización ya opera, y un piloto puede entrar en producción en días.

Azure AI Foundry: para ingenieros de ML y equipos de plataforma

Azure AI Foundry es para equipos que construyen agentes como producto. Proporciona el SDK completo, un catálogo de modelos, orquestación multi-agente, evaluación y observabilidad, que es lo que un equipo de plataforma necesita para operar agentes verticales de nivel productivo a escala. El catálogo de modelos

permite elegir un modelo por tarea y cambiarlo bajo control de versiones. La orquestación multi-agente coordina agentes especializados bajo un único runtime, con reintentos y tiempos de espera gestionados por la plataforma en lugar de escritos a mano en cada agente.

La gobernanza usa Azure RBAC y Azure AI Content Safety, los mismos controles de identidad y seguridad que gobiernan el resto de un entorno Azure. La evaluación y la observabilidad son de primera clase: el equipo define un conjunto de pruebas de referencia, corre regresión en cada cambio de modelo y traza cada decisión del agente para auditoría. El tiempo hasta un piloto funcional se mide en semanas, no en días, porque la preparación es más pesada y la vara de revisión es más alta. El retorno es un techo que los caminos low-code y de IDE no alcanzan, y por eso las cargas de producción reguladas caen aquí.

La matriz de decisión, lado a lado

Las tres plataformas son complementarias, no competitivas, y la mayoría de las empresas termina utilizando las tres. La pregunta abierta es qué caso de uso va a dónde, y seis ejes resuelven la mayor parte. Audiencia: Copilot Studio atiende a usuarios de negocio y desarrolladores ciudadanos, GitHub Copilot atiende a ingenieros, Azure AI Foundry atiende a ingenieros de ML y equipos de plataforma. Nivel de código: no-code y low-code, code-first en el IDE, y pro-code con SDK. Conectores: Microsoft 365 y Power Platform, repositorios y MCP, y cualquier fuente vía Python, REST o MCP.

Cómo leer la matriz por eje

Quedan tres ejes. El tiempo hasta el piloto va de horas a días, a días, a semanas, en ese orden. La personalización va de tópicos más conectores, a configuración de repositorio y herramientas MCP, a código completo más fine-tuning. La gobernanza va del centro de administración de Power Platform, a políticas de GitHub, a Azure RBAC con Content Safety. Lee los seis ejes en conjunto y la elección rara vez queda ambigua. Combina la plataforma con quién es dueño del agente y con la gobernanza que la carga exige, y deja que el tiempo hasta el piloto y la personalización desempaten lo que reste.

DEFINICIÓN

Studio para citizens, Copilot para ingenieros, Foundry para producción. Combina la plataforma con quién es dueño del agente y con la gobernanza que la carga exige, y deja que el tiempo hasta el piloto y la personalización desempaten.

Referencias

Fuentes primarias para las tres plataformas y el contexto de mercado detrás de la decisión de enrutamiento.

- Microsoft Copilot Studio documentation, la referencia para tópicos por arrastrar y soltar, conectores Microsoft 365 y Power Platform, y gobernanza por el centro de administración.
- GitHub Copilot documentation, la referencia para agent mode, custom instructions y herramientas del Model Context Protocol.
- Azure AI Foundry documentation, la referencia para el catálogo de modelos, orquestación multi-agente, evaluación y observabilidad.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, el contexto de mercado: 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

02 TIPOS DE AGENTES

Cuatro tipos de agentes y dónde encaja cada uno.

La elección de plataforma responde dónde corre un agente. El tipo de agente responde qué trabajo hace. La mayoría de los programas corporativos descubre los mismos cuatro tipos, y cada uno encaja en una plataforma específica por una razón específica. Este capítulo mapea agentes de desarrollo, productividad, procesos de negocio y datos a la plataforma que sirve, y nombra el guardrail que cada tipo exige antes de llegar a producción.

Cuatro tipos, y por qué cada uno encaja donde encaja

Los cuatro tipos no son una taxonomía por sí misma. Se agrupan por quién es dueño del agente, qué sistemas toca y cuánto cuesta una acción incorrecta. Los agentes de desarrollo sirven a ingenieros dentro del repositorio. Los agentes de productividad sirven a trabajadores del conocimiento dentro de Microsoft 365. Los agentes de procesos de negocio cruzan varios sistemas en nombre de un equipo de operaciones. Los agentes de datos y analítica responden preguntas contra un data warehouse. Una vez que sabes qué tipo estás construyendo, la elección de plataforma del capítulo anterior prácticamente se resuelve sola.

Gartner proyecta que el 40% de las aplicaciones corporativas tendrán agentes de IA task-specific para 2026, frente a menos del 5% en 2025. Task-specific es la expresión operativa. Los agentes que llegan a producción son estrechos. Hacen un tipo de trabajo, para una audiencia, con una superficie de control. Los cuatro tipos de abajo son las cuatro formas estrechas que se repiten entre los programas. Trátalos como zonas de aterrizaje, no como una frontera rígida. Un solo programa suele correr los cuatro lado a lado.

Agentes de desarrollo: el repositorio y el IDE

Los agentes de desarrollo viven donde los ingenieros ya trabajan: el repositorio, el pull request y el editor. La corrección de errores, la refactorización y la generación de pruebas son las tareas canónicas. En la óptica de GitHub Copilot, esto corre a través del agent mode. El agente lee el .github/copilot-instructions.md para las convenciones del repositorio, planifica un cambio, edita archivos, corre el build y abre un pull request. El ingeniero permanece en el bucle en el pull request, que es donde el code review ya ocurre.

MCP es lo que hace útil a un agente de desarrollo más allá del editor. Los servidores Model Context Protocol conectan los sistemas que un cambio real toca: el sistema de build, el rastreador de issues, el pipeline de despliegue. Un agente que puede leer una verificación fallida, abrir el ticket vinculado y correlacionar ambos escribe una mejor corrección que uno que solo ve el diff. El modelo viene del Copilot picker. Claude Opus 4.8 o Claude Fable 5 para las refactorizaciones difíciles, Sonnet 5 o Haiku 4.5 para las ediciones de rutina, donde la latencia y el costo pesan más que la profundidad.

La facturación sigue al modelo

El trabajo de un agente de desarrollo se factura en GitHub AI Credits, donde un crédito equivale a un centavo de dólar desde el 1 de junio de 2026. Los modelos más baratos del Copilot picker cuestan menos créditos por solicitud. Enrutar las ediciones de rutina a Haiku 4.5 y reservar Opus 4.8 para los cambios que lo necesitan es una línea en la factura, no una cuestión de gusto. Por eso la elección de modelo y el tipo de agente se deciden juntos, no por separado.

Agentes de productividad: dentro de Microsoft 365

Los agentes de productividad sirven a trabajadores del conocimiento donde ya pasan el día: Teams, Outlook y SharePoint. El trabajo es notas de reunión, generación de borradores y búsqueda en el contenido que la persona tiene permiso para ver. Copilot Studio crea los tópicos personalizados y los conecta a Microsoft 365, Dataverse y Power Platform. El equipo de negocio es dueño del agente, y el

centro de administración de Power Platform lo gobierna. El tiempo hasta un piloto funcional es de horas a días, porque el desarrollador ciudadano nunca sale de la superficie low-code.

La restricción que define este tipo es la recuperación con alcance de identidad. Un agente de productividad solo puede exponer contenido que el usuario solicitante ya tiene permiso para abrir. Las notas de reunión resumen una llamada en la que el usuario participó. La búsqueda devuelve documentos que el usuario podría haber encontrado a mano. Esto no es una función de conveniencia. Es la línea entre un asistente útil y una fuga de datos, y se impone en la capa de plataforma, no en el prompt.

Agentes de procesos de negocio: a través de varios sistemas

Los agentes de procesos de negocio ejecutan los flujos que mantienen una operación en movimiento: incorporación de empleados, aprobaciones de compra, reclamaciones de seguros. Estos flujos cruzan varios sistemas, que es lo que los separa de los dos primeros tipos. Un agente de reclamaciones lee un documento, verifica una póliza, actualiza un registro y notifica a una persona, cada paso en un sistema diferente. Copilot Studio puede llevar el frente conversacional de este trabajo. Alcanza su límite cuando el flujo necesita SLA estrictos, orquestación multi-agente o un modelo con fine-tuning.

Cuando lo hace, el flujo migra a Azure AI Foundry. Foundry le da al equipo de plataforma el SDK completo, orquestación multi-agente, harnesses de evaluación y Azure RBAC con Content Safety en cada llamada. La razón para migrar no es ambición. Es el SLA. Una aprobación que debe completarse dentro de un tiempo acotado, con un registro de auditoría en cada paso, necesita la observabilidad y la gobernanza que una superficie low-code no ofrece. El traspaso de Copilot Studio a Foundry es una etapa normal en un programa que madura, no una reescritura.

Agentes de datos y analítica: anclados en el data warehouse

Los agentes de datos y analítica convierten una pregunta en lenguaje natural en una consulta, la ejecutan y devuelven un resumen. Traducir una solicitud a SQL, generar un insight, señalar una anomalía. Azure AI Foundry es la plataforma, anclado en Fabric, Synapse o el data warehouse que la organización ya corre. El anclaje es el punto central. Un agente de analítica que infiere un esquema en lugar de leerlo devuelve números que parecen correctos y están equivocados.

Dos guardrails son obligatorios para este tipo. El primero es un harness de evaluación estructurado: un conjunto fijo de preguntas con respuestas conocidas que el agente debe pasar antes de cada cambio de modelo, porque una consulta que devuelve números plausibles pero incorrectos es peor que una que falla en voz alta. El segundo es la aprobación humana antes de publicar. El agente redacta el informe. Una persona lo firma. El costo de un número equivocado en una presentación de board se mide en decisiones, no en tokens.

> DEFINICIÓN
>
> El patrón en los cuatro tipos es el mismo. Más autonomía exige más guardrails, no menos. Un agente de desarrollo se detiene en el pull request. Un agente de productividad se detiene en los permisos del propio usuario. Un agente de procesos de negocio se detiene en el SLA y el registro de auditoría. Un agente de datos se detiene en la aprobación humana antes de publicar. El guardrail no es fricción. Es lo que permite que el agente corra.

Referencias

Fuentes primarias para los cuatro tipos de agentes, las plataformas donde encaja cada uno y la proyección de mercado detrás de ellos.

- GitHub Copilot documentation, agent mode, instrucciones de repositorio y MCP para agentes de desarrollo.

- Microsoft Copilot Studio documentation, la plataforma low-code para agentes de productividad y de procesos de negocio conversacionales.
- Azure AI Foundry documentation, la plataforma pro-code para agentes de procesos de negocio y de datos bajo SLA estrictos.
- 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, la proyección de mercado: los agentes task-specific pasan de menos del 5% de las aplicaciones corporativas en 2025 al 40% 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 ARQUITECTURA DE REFERENCIA

La arquitectura de referencia en seis capas: la pila de agentes enterprise.

Los programas de agentes de nivel productivo se construyen sobre los primitivos de la plataforma, no alrededor de ellos. La pila tiene seis capas, de la plataforma y gobernanza en la base hasta la superficie donde los usuarios encuentran al agente. Este capítulo recorre las seis capas, enuncia los cinco principios que rigen todas ellas, presenta tres patrones de orquestación y nombra las cuatro reglas de diseño que sostienen cada capa.

Seis capas, de la plataforma y gobernanza a la superficie

La pila se lee de abajo hacia arriba, y cada capa depende de la que está debajo. La base es la plataforma y la gobernanza: identidad, RBAC, Azure AI Content Safety, auditoría, observabilidad y FinOps, los mismos controles que ya rigen el resto de Azure. Dentro de GitHub Copilot, el costo en esta capa se mide en GitHub AI Credits, donde un crédito equivale a un centavo de dólar, facturado desde el 1 de junio de 2026, de modo que FinOps es una partida desde el primer piloto, no una ocurrencia tardía. La segunda capa son los modelos. El catálogo de Azure AI Foundry guarda los modelos alojados para los agentes de Foundry; para el trabajo de agente conducido desde 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. Elige por tarea y haz fine-tuning solo donde el retorno paga el pipeline. La tercera capa es el contexto: un vector store, una capa de contexto semántico, un registro de recetas y recuperación con alcance de identidad, de modo que un agente vea solo los datos que su llamador tiene permitido ver.

La cuarta capa son las herramientas, expuestas mediante el Model Context Protocol. Cada herramienta está catalogada, versionada y con alcance: servidores MCP, plugins nativos y skills personalizadas, cada una con un token que concede el

acceso más estrecho que la herramienta necesita. La quinta capa es la orquestación: el runtime multi-agente, la planificación, los reintentos, los puntos de control de Human-in-the-Loop y la evaluación. Es la capa que convierte una sola llamada de modelo en una secuencia de pasos que un revisor puede seguir. La sexta capa es la superficie: Teams, Outlook, el IDE y aplicaciones personalizadas. Es donde los usuarios encuentran al agente, y es la única capa que ven. Una brecha en cualquier capa inferior aparece aquí como una respuesta incorrecta, un registro filtrado o una acción que nadie aprobó.

Cinco principios que rigen cada capa

Cinco principios valen para las seis capas. Comienza estrecho: un caso de uso, un agente, un equipo, y generaliza solo después de probar el valor, porque un agente que hace un trabajo bien vale más que un framework que hace diez trabajos mal. Reutiliza la gobernanza de Azure en lugar de inventar un plano de control paralelo: la identidad, el RBAC y el Content Safety que protegen el resto del parque protegen también a los agentes, y un segundo modelo de gobernanza es una segunda cosa que auditar. Cataloga cada herramienta con un token de alcance y auditoría por llamada: una herramienta no catalogada es una herramienta no gobernada, y un token con más acceso del que la herramienta necesita es el radio de daño del próximo incidente.

Evalúa antes de desplegar: mantén un conjunto de pruebas de referencia y ejecuta una regresión en cada cambio de modelo, porque mover un agente de Sonnet 5 a Opus 4.8 cambia el comportamiento aun cuando el prompt es idéntico. La evaluación se ejecuta en el agent mode de GitHub Copilot contra el mismo .github/copilot-instructions.md con el que el agente se entrega, para que las condiciones de prueba coincidan con producción. Planifica la depreciación desde el primer día: versiona cada prompt, mantén disponible el rollback y trata un prompt como cualquier otro artefacto desplegable. Un programa de agentes que no puede revertir un cambio de prompt está a una mala edición de una caída que no sabe explicar.

Tres patrones de orquestación

La quinta capa ejecuta uno de tres patrones de orquestación, y la elección sigue la forma del trabajo. El pipeline secuencial encadena agentes en un orden fijo: cada agente consume la salida del agente anterior, y los traspasos son los puntos de control. Sirve para trabajo con etapas claras, como leer un documento, validarlo y luego ensamblar un resultado, donde cada etapa debe terminar antes de que empiece la siguiente. El patrón de fan-out y fan-in paralelo divide una tarea entre varios agentes que se ejecutan al mismo tiempo y luego combina sus salidas. Sirve para trabajo que se descompone en subtareas independientes, como puntuar a un solicitante contra varias listas de sanciones, donde las subtareas no dependen entre sí y el paso de combinación reconcilia los resultados.

El patrón de supervisor jerárquico con workers pone a un agente supervisor al mando de un grupo de agentes worker. El supervisor planifica, delega y decide cuándo se alcanzó el objetivo; los workers ejecutan tareas estrechas y reportan de vuelta. Sirve para trabajo abierto donde el plan no se conoce de antemano, como una tarea de código que puede tocar tres archivos o treinta. El supervisor también es el lugar natural para el punto de control de Human-in-the-Loop: sostiene el plan, así que puede pausar para aprobación antes de que un worker realice una acción irreversible. En el agent mode de GitHub Copilot, ese es el patrón detrás de un coding agent que lee el repositorio, redacta un cambio en varios archivos y se detiene en un gate de revisión antes de abrir un pull request.

Cuatro reglas de diseño que sostienen cada capa

Cuatro reglas de diseño mantienen sólida la pila a medida que crece. Responsabilidad única: cada agente es dueño de un trabajo, y una tarea que necesita tres capacidades son tres agentes detrás de un supervisor, no un agente con tres prompts, porque un agente de responsabilidad única es uno que puedes probar, costear y explicar. Predeterminado a prueba de fallos: cuando un agente está incierto o una llamada de herramienta falla, el valor por defecto es detenerse y escalar a un humano, nunca adivinar y continuar. El costo de una acción autónoma incorrecta sobre datos regulados supera el ahorro de cien acciones correctas, así que el camino seguro es el camino por defecto.

Auditoría inmutable: cada llamada de herramienta, elección de modelo y decisión de Human-in-the-Loop queda registrada en un registro que ningún agente puede alterar, para que un revisor pueda reconstruir exactamente lo que cualquier agente vio e hizo en cualquier momento. Degradación elegante: cuando una capa inferior no está disponible, el agente estrecha su alcance en lugar de fallar por completo, de modo que un contexto que queda brevemente sin conexión rebaja al agente a un conjunto de acciones menor y más seguro en lugar de derribarlo. Juntas, las cuatro reglas significan que una falla en una capa permanece contenida en esa capa en lugar de cascadear hasta la superficie.

> DEFINICIÓN
>
> La pila en una frase: seis capas, de la plataforma y gobernanza hasta la superficie; cinco principios que comienzan estrechos y planifican la depreciación; tres patrones de orquestación combinados con la forma del trabajo; y cuatro reglas de diseño, responsabilidad única, predeterminado a prueba de fallos, auditoría inmutable y degradación elegante, que impiden que una falla en una capa llegue al usuario.

Referencias

Fuentes primarias para las capas de plataforma, el catálogo de modelos y la tendencia de adopción enterprise.

- Azure AI Foundry documentation, el catálogo de modelos, la orquestación y la evaluación detrás de las capas de modelos y orquestación.
- Microsoft Copilot Studio documentation, la superficie low-code y la gobernanza para agentes propiedad del negocio.
- GitHub Copilot documentation, la plataforma de referencia para agent mode, copilot-instructions.md e integración MCP.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, la tendencia de adopción enterprise detrás de la pila de seis capas: por encima de 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

04 SUPERVISIÓN HUMANA

Supervisión humana y los guardrails que escalan con la autonomía.

La autonomía no es el objetivo. Los resultados confiables lo son. Un agente que puede actuar sobre datos regulados, firmar documentos, mover dinero o modificar registros de clientes necesita puntos de control humanos explícitos, porque una acción autónoma incorrecta puede costar más de lo que ahorran cien acciones correctas. Este capítulo enuncia el principio rector, ordena cada acción del agente en cuatro cuadrantes, presenta tres patrones de implementación para el punto de control y define la base de seguridad que queda por debajo de ellos.

El principio rector: más autonomía exige más guardrails

La regla es simple, y va contra la intuición. Más autonomía exige más guardrails, no menos. Un agente que solo sugiere texto necesita poca supervisión. Un agente que puede actuar sobre datos regulados, firmar documentos, mover dinero o modificar registros de clientes necesita puntos de control humanos explícitos. La razón es la asimetría. Una acción autónoma incorrecta puede costar más de lo que ahorran cien acciones correctas. Un reembolso enviado a la cuenta equivocada, un contrato firmado en malas condiciones, una tabla de producción eliminada: cada uno de estos es caro de revertir, y algunos no se pueden revertir en absoluto. Los guardrails existen para impedir que la rara acción catastrófica llegue a producción sin supervisión.

Este principio se sostiene sin importar qué modelo ejecuta el agente. En GitHub Copilot en agent mode eliges el modelo en el Copilot picker: Claude Fable 5, Opus 4.8, Sonnet 5, Haiku 4.5, Gemini 3.1 Pro o un modelo GPT-5.x. Un modelo más fuerte planifica mejor y actúa más rápido. No reduce el costo de una acción incorrecta. Si algo cambia, un agente más rápido amplía el radio de impacto, porque alcanza el

paso irreversible antes. Así que los guardrails escalan con la capacidad, no contra ella. El lugar para declararlos es .github/copilot-instructions.md, el archivo que el agente lee en cada tarea. Lleva las convenciones, los límites y la regla de cuándo detenerse y preguntar a un humano.

> DEFINICIÓN
>
> La regla que rige la supervisión humana: más autonomía exige más guardrails, no menos. Una acción autónoma incorrecta puede costar más de lo que ahorran cien acciones correctas, así que el guardrail escala con el radio de impacto, no con la confianza del modelo.

Cuatro cuadrantes: impacto contra reversibilidad

No toda acción necesita la misma supervisión. Ordena cada acción que un agente puede tomar en dos ejes: impacto en el negocio y reversibilidad. Eso produce cuatro cuadrantes, y cada uno recibe un nivel distinto de supervisión. El cuadrante uno es automatizar: bajo impacto, alta reversibilidad. Sugerencias de mensajes de commit, resúmenes de borradores de correo, notas de reunión. Corren dentro del loop del agente con registro de auditoría y sin punto de control, porque un error es barato y fácil de deshacer. El cuadrante dos es revisar: alto impacto, alta reversibilidad. Abrir un pull request, programar una reunión, enviar un borrador a un cliente. Un humano aprueba antes del commit o del envío, y la acción todavía se puede deshacer si se escapa.

El cuadrante tres es confirmar: bajo impacto, baja reversibilidad. Archivar un ticket, cerrar una conversación, eliminar un archivo temporal. El daño es pequeño pero permanente, así que un único paso de confirmación basta. El cuadrante cuatro es control dual: alto impacto, baja reversibilidad. Firmar un contrato, transferir fondos, cambiar permisos RBAC, eliminar datos de producción, presentar impuestos. Dos personas aprueban, y el registro de auditoría lleva evidencia firmada. La mayoría de las acciones de agentes empresariales pertenece al cuadrante dos o tres, no al cuadrante uno. GitHub Copilot en agent mode encaja directamente en este modelo: el agente propone su trabajo como un pull request, y el pull request es, en sí mismo,

el punto de control del cuadrante dos, donde una persona revisa el diff antes del merge.

Tres patrones de implementación para el punto de control

Tres patrones cubren la mayor parte de lo que los cuadrantes exigen. Aprobar antes es el patrón para el trabajo reversible de alto impacto. El agente prepara una acción y espera en un punto de control. Nada se ejecuta hasta que un humano aprueba, y la aprobación se registra con la identidad del revisor. En agent mode esto es la revisión del pull request: el agente escribe el cambio, una persona lee el diff, y el merge es la aprobación. Muestreo y revisión sirve para el trabajo de alto volumen y bajo impacto, como el etiquetado, la clasificación y el enrutamiento. El agente actúa de forma continua, una muestra aleatoria de sus acciones se audita, y una caída en la métrica de calidad muestreada dispara el escalado. No revisas cada acción. Revisas lo suficiente para confiar en la distribución.

Control dual es para el cuadrante irreversible. Una persona inicia, una persona separada aprueba, y las dos nunca son la misma identidad. Ambas aprobaciones llevan firma criptográfica, y el par se almacena como evidencia. Es obligatorio para finanzas, jurídico y cualquier cambio en datos de producción. Los tres patrones también moldean el costo. En GitHub Copilot, cada acción del agente se factura en GitHub AI Credits, donde un crédito equivale a un centavo de dólar, el modelo de facturación en vigor desde el 1 de junio de 2026. Muestreo y revisión mantiene el riesgo y el gasto observables en alto volumen, porque miras los resultados muestreados y los AI Credits que consumen en la misma vista. Las herramientas llegan al agente por servidores MCP, y cada servidor debe exponer solo las operaciones que la tarea necesita.

Seguridad y gobernanza: de least privilege a la retención de siete años

Los guardrails al nivel de la acción necesitan una base de seguridad por debajo. Least privilege va primero. Un agente, y cada servidor MCP que llama, recibe acceso solo a los sistemas que la tarea específica requiere, y nada más. Un agente de

código no necesita la API de pagos. Un agente de triaje no necesita acceso de escritura a producción. La autenticación usa Azure Managed Identity, nunca una cadena de conexión hardcoded en un prompt, un repositorio o un archivo de entorno. Managed identity significa que la credencial la emite y la rota la plataforma, y nunca aparece en texto que el agente pueda filtrar. Alcance estrecho y managed identity, juntos, eliminan las formas más comunes en que una acción autónoma se convierte en un incidente.

El rastro de auditoría es la otra mitad. Cada acción del agente se escribe en un registro inmutable: qué agente, qué versión, qué acción, la entrada y la salida como hashes en lugar de dato crudo, el humano que la aprobó y la duración. Los datos personales se enmascaran antes de llegar al registro. Un identificador como un CPF o una dirección se enmascara para que el registro pruebe qué pasó sin exponer a quién le pasó. La retención es larga. Las cargas reguladas en Brasil mantienen el registro de auditoría por 2555 días, siete años, para coincidir con el plazo fiscal de retención, guardado en Azure Immutable Storage para que nadie pueda reescribir la historia después. La política que el agente sigue vive en .github/copilot-instructions.md, así que el mismo archivo que define las convenciones también codifica el límite que el agente no debe cruzar.

Referencias

Fuentes primarias para el modelo de supervisión, los patrones de guardrail y la base de seguridad y gobernanza.

- GitHub Copilot documentation, la plataforma de referencia para agent mode, .github/copilot-instructions.md e integración de herramientas MCP.
- Azure AI Foundry documentation, gobernanza para agentes de producción mediante Azure RBAC, managed identity y Azure AI Content Safety.
- Microsoft Copilot Studio documentation, gobernanza para agentes propiedad del negocio mediante el centro de administración de Power Platform.
- Azure, Responsible use of AI overview, guía de Microsoft sobre supervisión humana y content safety para sistemas de IA.
- LGPD, Lei Geral de Protecao de Dados, la base legal para enmascarar datos personales en registros de auditoría en cargas reguladas en Brasil.

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 PARTE V, ENTREGAR Y ESCALAR

Del primer piloto a una flota de agentes en producción.

Un piloto prueba que un agente funciona una vez. Una flota prueba que funciona todos los días, sobre datos regulados, bajo auditoría, en equipos que nunca conocieron a quienes construyeron el primero. La distancia entre esos dos hechos es donde la mayoría de los programas de agentes se estanca. Este capítulo la recorre con un caso de uso trabajado, cuatro patrones verticales que reutilizan su coreografía, y una hoja de ruta de doce meses cuyos gates de fase mantienen el programa honesto.

El caso de leasing de vehículos: cuatro agentes encadenados bajo gates Human-in-the-Loop

Empieza con un flujo lo bastante estrecho para terminar y lo bastante real para importar. Un cliente solicita el leasing de un vehículo. Cuatro agentes especializados llevan la solicitud de extremo a extremo, cada uno con una función. El agente de intake recoge la solicitud y normaliza los campos. El agente de parsing de documentos lee los archivos cargados: comprobante de ingresos, identidad, domicilio. El agente de validación de NF-e coteja la factura electrónica, el estándar brasileño para documentos fiscales, contra el registro tributario que dice corresponder. El agente de ensamblaje de contrato redacta el leasing a partir de las entradas aprobadas. Ningún agente es dueño del flujo completo. Cada uno entrega un resultado tipado al siguiente, y eso es lo que hace la cadena auditable.

Los gates Human-in-the-Loop se ubican entre los pasos, colocados por reversibilidad e impacto. El parsing de documentos corre en la banda de automatización con registro de auditoría. La validación de NF-e es un gate de revisión, porque una correspondencia tributaria errónea es cara y lenta de deshacer. La firma del contrato es control dual: un iniciador y un aprobador separado, nunca la misma persona. Construido sobre GitHub Copilot, cada agente es un agente

personalizado con alcance definido vía .github/copilot-instructions.md, con servidores MCP que conectan el repositorio de documentos, la consulta tributaria y el sistema de contratos. La elección de modelo viene del selector de Copilot en cada paso: Haiku 4.5 para clasificación barata, Opus 4.8 para el razonamiento de contrato que tiene que estar bien. Cada llamada se factura en GitHub AI Credits, donde un crédito es un centavo de dólar, así que el costo por solicitud es un número que lees, no una conjetura.

Patrones verticales que reutilizan la misma coreografía

La cadena de leasing no es un caso aislado. La misma coreografía, agentes especializados en secuencia con gates humanos en los puntos de alto impacto, se transfiere a cualquier flujo regulado y pesado en documentos. En finanzas, el triaje de KYC y AML corre un agente de documentos que lee archivos de incorporación, un agente de puntuación de riesgo que cruza nombres con listas de sanciones, y un agente de escalado que dirige al responsable de cumplimiento, con control dual en la aprobación. En salud, la autorización previa combina un agente de extracción clínica con una capa de recuperación de reglas de pagadores y un punto de control del médico antes de la presentación. El tiempo de autorización baja de días a horas, y cada decisión queda registrada para auditoría.

El retail corre el mismo formato para devoluciones y logística inversa. Un agente de triaje clasifica el motivo de la devolución, un agente de reembolso ejecuta la verificación de política, y un agente de logística programa la recogida. El gate humano se activa solo en excepciones por encima del umbral de SKU, así que las devoluciones de rutina pasan sin revisor. La manufactura lo aplica al mantenimiento predictivo: un agente de telemetría vigila las señales de IoT, un agente de diagnóstico propone órdenes de trabajo, y un agente de programación asigna a los técnicos. La aprobación del gerente de planta es necesaria para cualquier orden de trabajo por encima de cuatro horas.

La coreografía es portable; las reglas de dominio no.

Lo que atraviesa las verticales es el patrón: agentes estrechos, entregas tipadas, gates colocados por impacto. Lo que no atraviesa es el contenido de dominio, la lista

de sanciones, las reglas de pagadores, la política de SKU, los umbrales de mantenimiento. Mantén ese contenido en archivos de instrucción versionados y herramientas MCP, no en prompts, y la flota de leasing de un equipo se convierte en la flota de KYC del siguiente en semanas, no en trimestres.

La hoja de ruta de cinco fases y doce meses

Una flota se construye en cinco fases a lo largo de doce meses, no en un sprint heroico. Descubrir ocupa el primer mes: mapear casos de uso candidatos, puntuar cada uno por valor versus esfuerzo, y obtener la aprobación de los interesados antes de que alguien escriba un agente. Pilotar corre en los meses dos y tres: un agente, un equipo, un harness de evaluación en bucle cerrado que mide si el agente de verdad ayuda. Fortalecer llena los meses cuatro a seis: RBAC y auditoría cableados, SLA y límites de costo fijados en GitHub AI Credits, y los puntos de control Human-in-the-Loop definidos para cada acción que el agente puede ejecutar.

Escalar corre de los meses siete a nueve: un registro de herramientas versionado, los patrones de coreografía multi-agente del caso de uso hechos reutilizables, e incorporación de autoservicio para que un equipo nuevo levante su propia flota sin un ingeniero de plataforma en la sala. Operar son los meses diez a doce y todo lo que sigue: FinOps sobre el gasto de AI Credits, gestión de deriva, ejercicios de cambio de modelo que reejecutan el harness de evaluación cuando un nuevo modelo llega al selector de Copilot, y una cadencia de revisión trimestral. Las fases son acumulativas. Cada una supone que la anterior se cerró.

Criterios de salida por fase

Cada fase termina en un criterio, no en una fecha. Descubrir sale cuando la lista corta está aprobada y el caso de uso principal tiene una métrica de éxito medible. Pilotar sale cuando el harness de evaluación muestra al agente alcanzando esa métrica en casos reservados, no en la demo. Fortalecer sale cuando un auditor puede rastrear cualquier acción del agente hasta una identidad y una política. Escalar sale cuando un equipo que no entrenaste se incorpora solo. Operar no tiene

salida; es el estado estable, y su criterio es que la revisión trimestral siga encontrando el programa sano.

Los criterios existen porque las dos fallas más comunes son saltos de fase. Pilotar sin fortalecer produce una demostración: funciona en la sala y se cae la primera vez que encuentra datos reales y un auditor real. Escalar sin operar produce deuda: decenas de agentes en producción sin nadie vigilando el gasto, la deriva ni los cambios de modelo por debajo. Un gate de fase es una afirmación que puedes defender en una revisión de board. Trátalo así y la flota llega a tiempo y se mantiene en pie.

> DEFINICIÓN
>
> Pilotar sin fortalecer es una demostración. Escalar sin operar es deuda. Cada gate de fase es una afirmación que puedes defender en una revisión de board, no una fecha en un slide.

Referencias

Fuentes primarias para las elecciones de plataforma, la hoja de ruta de producción y el contexto de mercado detrás de una flota de agentes enterprise.

- GitHub Copilot documentation, el camino de ingeniería usado para construir los agentes del caso de uso: agent mode, agentes personalizados y MCP.
- Azure AI Foundry documentation, la plataforma de producción para las fases de fortalecer, escalar y operar: catálogo de modelos, orquestación multi-agente, evaluación y observabilidad.
- Microsoft Copilot Studio documentation, el camino low-code para agentes propiedad del negocio en las fases de descubrir y pilotar.
- Gartner, 40% of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, contexto de mercado para por qué importa la flota: agentes de IA específicos por tarea proyectados al 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