3HTP CLOUD SERVICES



Tu agente falla. Y no es culpa del modelo.

La ingeniería que convierte un LLM en un operador confiable

www.3htp.com

Resumen

Pagaste por el modelo más caro del mercado. Lo cambiaste tres veces. Probaste el de un proveedor, luego el de otro, volviste al primero. Y tu agente sigue fallando de maneras que no puedes explicarle a tu comité de dirección: ejecuta una acción que no debía, pierde el hilo a mitad de una tarea, o funciona perfecto en la demo y se desmorona en producción.

El modelo, casi siempre, hizo exactamente lo que le pediste. **El problema es que nadie diseñó lo que lo rodea.** Esa capa —contexto, herramientas, memoria, orquestación, permisos, verificación y observabilidad— se ha consolidado como término de industria en 2025–2026: el **harness**. Y la tesis de este artículo es incómoda pero defendible: *dado un modelo de frontera suficientemente capaz, el harness es la variable dominante en un despliegue empresarial confiable.*

El modelo razona. El harness responde por lo que ese razonamiento hace en tus sistemas.

Estructura del artículo

- El problema: por qué el mismo modelo rinde distinto
- Qué es el harness (y qué no es)
- Anatomía: los componentes de un harness
- El paisaje: implementaciones gestionadas (agnóstico de nube)
- Gobernanza: el harness como plano de control
- Qué queda cuando los modelos absorban el harness
- Un caso real + recomendación 3HTP

A quién va dirigido: CTOs, arquitectos de soluciones y líderes de plataforma que están llevando IA agéntica a producción —agentes con acceso de escritura a sistemas reales— y necesitan entender por qué sus agentes brillan en la demo y fallan en operación.

Prerrequisitos asumidos: familiaridad básica con agentes de IA (un modelo que decide qué herramientas usar, en qué orden y cuándo parar) y con conceptos de arquitectura de software.

El problema: el mismo modelo, resultados distintos

Durante tres años la conversación fue sobre el modelo. Más parámetros, más contexto, mejores benchmarks. Y es cierto que el modelo importa —importa muchísimo—: un modelo que resuelve el 10 % de las tareas de un benchmark de ingeniería y otro que resuelve el 50 % no son intercambiables, y **ningún harness compensa esa diferencia de capacidad base**. Conviene decirlo de entrada, porque la tesis de este artículo no es "el modelo da igual". Es otra: *pasado cierto umbral de capacidad, lo que separa un agente que funciona en producción de uno que no, ya no es el modelo.*

La evidencia de que el rendimiento de un agente es un fenómeno de **sistema**, no de modelo aislado, viene del propio equipo de ingeniería de Anthropic. Al analizar su sistema multiagente de investigación sobre la evaluación BrowseComp, documentaron que **tres factores explicaban el 95 % de la varianza de rendimiento; el uso de tokens por sí solo explicaba el 80 %** (Anthropic, 2025). Los tres factores eran: el número de tokens gastados, el número de llamadas a herramientas y la elección de modelo.

Léelo con cuidado, sin sobreinterpretar. La elección de modelo sí está en esa ecuación; es uno de los tres factores. Lo que el dato muestra no es que el modelo no importe, sino algo más preciso: **el rendimiento del agente emerge del diseño del sistema completo**. Anthropic lo atribuye a la arquitectura: repartir el trabajo entre agentes con contextos separados es lo que multiplica el gasto de tokens y, con él, la capacidad de razonamiento. Esas decisiones de reparto, orquestación y herramientas pertenecen al harness, no al modelo.

Qué es el harness (y qué no es)

Un **harness** es el conjunto de herramientas, contexto, memoria, permisos, bucles de verificación y entorno de ejecución que envuelve a un modelo de lenguaje para convertirlo en un agente capaz de hacer trabajo real. No es el modelo. Es todo lo demás.

La formulación que ha adoptado la comunidad es una ecuación simple:

> **Agente = Modelo + Harness**

Microsoft lo define como el "andamiaje de ejecución" que convierte un modelo en un agente que realiza trabajo: dirige las llamadas al modelo y a las herramientas, administra el estado y el contexto, aplica políticas de aprobación y mantiene el progreso en tareas de varios pasos (Microsoft, s. f.).

Hay un matiz de uso. Hugging Face distingue el harness, que es el bucle de ejecución que llama al modelo y gestiona sus herramientas, del *scaffolding*, que son las instrucciones, herramientas y formato con los que trabaja el modelo (Hugging Face, s. f.). Aquí usamos el término en sentido amplio, como lo hacen los productos que se presentan a sí mismos como harness.

La palabra no es casual. *Harness* viene de la equitación: las riendas, la silla, el bocado —el equipamiento que canaliza la fuerza de un animal poderoso pero impredecible en la dirección correcta. Si prefieres una imagen mecánica: **el modelo es el motor; el harness es el resto del coche.** El chasis, el volante, los frenos, el parabrisas, los testigos del tablero. Un motor de Fórmula 1 sin frenos no es rápido: es peligroso.

Qué NO es un harness. No es un prompt. Un prompt es una instrucción de texto; el harness es la infraestructura que permite al modelo *actuar* sobre esa instrucción. Tampoco es exactamente lo mismo que un *framework* o un *runtime* de agentes, y la distinción importa. Harrison Chase

(2025), de LangChain, propuso esta taxonomía, y reconoce que no fue él quien acuñó el término "harness":

Capa	Rol	Ejemplo en el ecosistema LangChain
Framework	Abstracciones componibles para construir agentes	LangChain
Runtime	Sustrato de ejecución: persistencia, ejecución durable, aprobación humana	LangGraph
Harness	Capa integrada y con criterio propio: planificación, archivos, subagentes, memoria	Deep Agents

La idea de que esto es una disciplina, y no un término de moda, tiene respaldo académico creciente. El trabajo *From Model Scaling to System Scaling* (Gu, 2026) sostiene que el próximo cuello de botella de la IA agéntica no es escalar el modelo, sino **escalar el harness**: tratar la capa estructurada que rodea al modelo como un objeto de primera clase de diseño, evaluación y optimización. Grupos de Stanford (*Meta-Harness*) y de Google DeepMind (*AutoHarness*) publican métodos para buscar y sintetizar código de harness de forma automática (Lee et al., 2026; Lou et al., 2026).

Una forma útil de ubicar esta disciplina es verla como tres fases: **prompt engineering** (optimizar la instrucción) → **context engineering** (construir dinámicamente lo que el modelo ve) → **harness engineering** (diseñar el entorno completo donde el agente trabaja, con restricciones, verificaciones y bucles de mejora). Cada fase resuelve lo que la anterior no podía: un agente que ignora las dependencias entre módulos no deja de hacerlo con un mejor prompt; necesita una regla determinista que lo bloquee.

Por qué ahora

Tres señales convergen para que esto deje de ser un debate de laboratorio.

Las tareas autónomas se alargan. METR documentó que la duración de las tareas que los agentes de frontera completan con 50 % de fiabilidad se duplica aproximadamente **cada 7 meses** (METR, 2025). Ese cálculo se publicó en marzo de 2025, con datos hasta esa fecha. Anthropic, por su parte, midió que entre octubre de 2025 y enero de 2026 el percentil 99,9 de la duración de los turnos de Claude Code pasó de menos de 25 minutos a más de 45 minutos (Anthropic, 2026). Cuanto más larga la tarea autónoma, más depende el resultado de que el sistema sostenga ejecución, memoria y recuperación: es decir, del harness.

Los agentes se multiplican. Gartner proyecta que una empresa global promedio del Fortune 500 tendrá en uso más de **150.000 agentes** para 2028, frente a menos de 15 en 2025. Gartner lo llama *agent sprawl* (Gartner, 2026). Gobernar esa cantidad de agentes sin una capa común

de control, permisos y observabilidad no es un problema de modelo. Es un problema de harness.

Y cuando el harness falta, se rompe algo real. En 2026 se registró el CVE-2026-27966 sobre el componente *CSV Agent* de Langflow. El código tenía **hardcodeada** una herramienta de ejecución de código (`allow_dangerous_code=True`), lo que exponía automáticamente un intérprete de Python. Una inyección de prompt podía entonces encadenarse hasta ejecución remota de comandos en el servidor. El aviso de seguridad (GitHub, s. f.) indica que no existía ninguna opción de configuración para desactivar ese comportamiento. El matiz es exactamente el punto de este artículo: **el modelo no falló. El modelo ejecutó precisamente lo que el harness le permitía hacer.** Fallaron dos capas a la vez: la de **permisos** (una herramienta de altísimo riesgo quedó habilitada por defecto, sin forma de desactivarla) y la de **aislamiento** (el código se ejecutaba directamente en el servidor, sin un entorno que lo contuviera). No fue un fallo de inteligencia; fue un fallo de andamiaje.

Anatomía de un harness

Si abstraemos lo que comparten los harnesses de la industria, emergen ocho componentes. Piénsalos como una lista de verificación, no como una especificación cerrada: un agente pequeño necesita pocos; uno de producción, casi todos.

1. **Contexto.** Qué ve el modelo. Selección, compactación y ensamblado de la información relevante. Regla de oro: *si no está en el contexto, para el modelo no existe*. Meter todo el código de golpe no ayuda —degrada la precisión—; la habilidad está en darle las herramientas para que encuentre lo que necesita.
2. **Herramientas.** Cómo el agente actúa sobre el mundo: leer y escribir archivos, consultar bases de datos, llamar APIs, ejecutar código. El estándar emergente de integración es **MCP (Model Context Protocol)**, que permite conectar servidores de herramientas sin integraciones a medida.
3. **Memoria y estado.** Qué recuerda entre pasos y entre sesiones. Historial, resúmenes, listas de tareas, preferencias persistidas.
4. **Orquestación y bucle de control.** Planificación, descomposición en subtareas, decisión de cuándo parar, manejo de errores, coordinación de subagentes.
5. **Entorno de ejecución y aislamiento.** Dónde corre y actúa el agente: un sandbox, una microVM, un contenedor efímero. *La segunda línea de defensa ausente en el caso Langflow: un sandbox habría contenido el daño aunque la herramienta peligrosa estuviera habilitada.*
6. **Guardrails y permisos.** Las reglas: validación de entradas, validación de salidas, autorización por acción, aprobación humana para lo irreversible, y la **identidad del agente** (qué credenciales porta, cómo se autentica ante otros sistemas, cómo se le atribuyen sus acciones). Conviene no colapsar estas sub-capas: proteger solo la entrada deja la salida expuesta, y una herramienta de alto riesgo habilitada por defecto —como en el caso Langflow— es un fallo de esta capa, el de permisos.

7. **Verificación.** Comprobar la calidad del trabajo antes de aceptarlo. Es la diferencia entre "el agente casi siempre pasa el linter" y "el agente siempre pasa el linter".
8. **Observabilidad y trazabilidad.** Cuando una tarea de cincuenta pasos falla en el paso 37, una traza te dice qué pasó. Registra llamadas, herramientas, errores, latencia y coste.

> **Para el lector CTO:** mapea tus agentes contra estos ocho componentes. El que esté vacío es, con alta probabilidad, tu próximo punto de falla.

El paisaje: harnesses gestionados

Lo bueno de que el harness sea un patrón reconocible es que ya no hay que construirlo entero a mano. Las principales plataformas ofrecen harnesses gestionados que implementan buena parte de los ocho componentes. Los presentamos lado a lado, sin preferencia de proveedor:

Plataforma	Qué aporta como harness
Amazon Bedrock AgentCore (AWS)	Runtime con aislamiento por microVM, Gateway de herramientas (MCP), Memory, Identity, Observability, Evaluations y políticas en Cedar que condicionan cada llamada a herramientas.
Microsoft Agent Framework (.NET, Python y Go)	Bucle de agente con llamadas a herramientas, gestión de contexto y estado, y políticas de aprobación.
LangChain Deep Agents (sobre LangGraph)	Planificación, sistema de archivos virtual para el contexto, subagentes, memoria a largo plazo, compresión del historial y middleware de aprobación humana.
Agent SDK de Anthropic	Las mismas herramientas, bucle y gestión de contexto que usa Claude Code, como biblioteca en Python y TypeScript: herramientas integradas, hooks, subagentes, permisos y MCP.
Claude Managed Agents (Anthropic)	Harness alojado: ejecuta el bucle del agente en un entorno gestionado por Anthropic o en una sandbox propia.

Nota. Descripciones basadas en la documentación oficial de cada proveedor (Amazon Web Services, s. f.; Anthropic, s. f.; LangChain, s. f.; Microsoft, s. f.).

El patrón es el mismo en todas; cambian los detalles de implementación y el ecosistema. Elegir un harness empieza a ser una decisión **tan importante como —y separada de— elegir el modelo**.

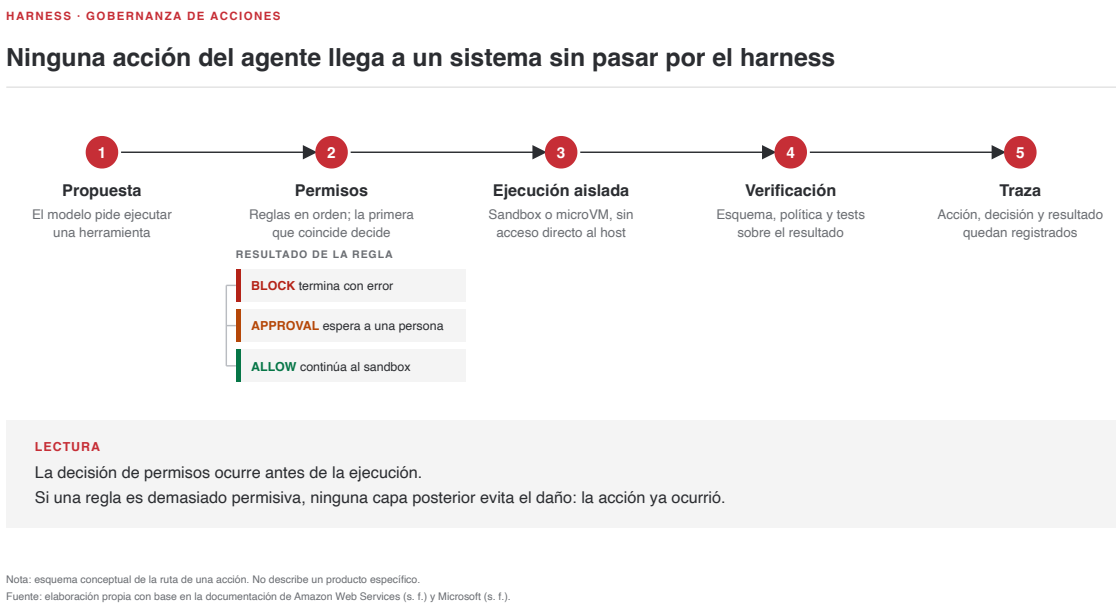
Gobernanza: el harness como plano de control

Aquí es donde el harness deja de ser una conveniencia de ingeniería y se vuelve el **plano de control de la seguridad y la gobernanza** de tus agentes. Es, además, donde se materializan tres discusiones que ya hemos abordado en esta serie —esta vez de forma deliberadamente agnóstica de nube, porque los principios del harness preceden a la elección de plataforma—:

- **La autonomía graduada por acción** (de copiloto a operador): decidir qué puede ejecutar el agente solo y qué requiere aprobación humana *se implementa en el harness* —en la capa de guardrails y permisos—, no en el modelo.
- **La defensa contra prompt injection**: el interceptor que evalúa el contenido antes de que el modelo lo procese *es un componente del harness*.
- **La seguridad de agentes en general** (identidad, mínimo privilegio, aislamiento): son capas del harness, no propiedades del modelo.

FIGURA 1

Ruta de una acción del agente a través del harness



Nota: esquema conceptual de la ruta de una acción. No describe un producto específico.
Fuente: elaboración propia con base en la documentación de Amazon Web Services (s. f.) y Microsoft (s. f.).

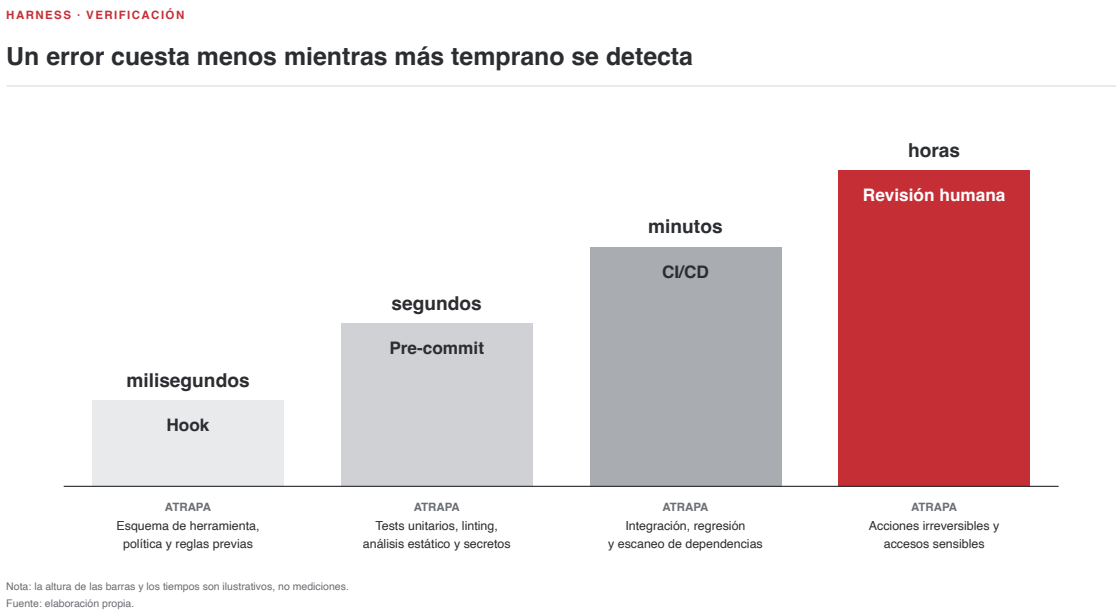
Nota. Esquema conceptual; no describe un producto específico. Elaboración propia.

Y aquí un matiz que un arquitecto no debe pasar por alto, y que la propia documentación de AWS hace explícito para AgentCore (Amazon Web Services, s. f.): **el harness valida la estructura de la petición, pero no inspecciona el significado de los prompts ni impone restricciones de conducta al agente.** En otras palabras: **la plataforma te da fronteras, no te exime de diseñar la seguridad.** La validación de entradas y la autorización del llamador siguen siendo tu responsabilidad.

La buena noticia es que la industria converge en cómo estructurar esos controles. Un patrón robusto es la **verificación en capas**, donde cada capa atrapa el error más barato posible antes de que escale:

FIGURA 2

Verificación en capas: cada nivel atrapa errores a un costo mayor



Nota. Alturas y tiempos ilustrativos, no mediciones. Elaboración propia.

Cuanto antes se detecta el error, más barato es corregirlo, y menos depende de la buena voluntad del modelo. Marcos de referencia distintos piden controles con lógica parecida. El *Well-Architected Agentic AI Lens* de AWS (Amazon Web Services, s. f.) está organizado por los seis pilares del framework, y el OWASP AISVS define requisitos verificables de seguridad para sistemas de IA (OWASP Foundation, s. f.). Que lleguen a principios similares es una señal de consenso, no de moda.

Lo incómodo: ¿qué queda cuando los modelos absorban el harness?

Un lector honesto hará la pregunta difícil: *si los modelos de frontera están absorbiendo capas del harness —tool use nativo, memoria integrada, contextos de un millón de tokens—, ¿no envejecerá mal todo esto?*

Es una objeción válida y hay que responderla de frente. La dirección de la industria, en efecto, es que los modelos integren progresivamente capacidades que antes vivían en el harness. Hace pocos años, la recuperación de documentos (RAG) y la memoria se construían siempre fuera del modelo; hoy algunas plataformas y modelos las ofrecen de serie.

Conviene entonces distinguir dos cosas:

- **El harness como producto** (un tooling concreto) — sí, es parcialmente absorbible. Partes de él migrarán al modelo o a la plataforma. Apostar a que tu *tooling* específico sea tu ventaja

permanente es frágil.

- **El harness como disciplina** (diseñar contexto, verificación, permisos y gobernanza para un propósito y un dominio concretos) — eso no lo absorbe ningún modelo.

La formulación honesta, por tanto, no es "el harness es la ventaja para siempre". Es: **en la ventana 2024–2027, mientras los modelos aún no absorben estas capas, el harness es donde se gana o se pierde la confiabilidad en producción.** Y lo más importante: **la disciplina de diseñar harnesses no desaparece cuando el modelo absorba las herramientas; migra hacia arriba en la pila.** Un modelo podrá traer memoria, recuperación y ejecución de serie; nunca traerá de fábrica el criterio de qué es irreversible en *tu* negocio, qué datos no pueden cruzarse entre *tus* clientes, ni qué significa "aprobación" en *tu* organización. Eso siempre habrá que diseñarlo.

Un caso real: nuestro propio harness (y la cicatriz que lo formó)

Lo justo, después de todo lo anterior, es mostrar un harness real en lugar de describirlo en abstracto. Usamos el nuestro —el framework de desarrollo con el que 3HTP opera sus propios proyectos— no como "el harness ideal", sino como *un* ejemplo de cómo estos principios se encarnan. Y empezamos por una cicatriz, porque un caso sin fallo no enseña nada.

La cicatriz. En una de sus primeras versiones, la capa de conocimiento del framework —la memoria que evita que el agente redescubra lo que ya sabe— se guardó entera en un directorio local no versionado. El resultado: cada miembro del equipo acumulaba memoria *en su propia máquina*, rompiendo justo el propósito de tener memoria *de equipo*. No fue un fallo del modelo; fue un fallo de diseño del harness. Se corrigió separando el conocimiento compartido (versionado, revisable por pull request) de los índices regenerables (locales), y quedó registrado como decisión de arquitectura. El harness, literalmente, **aprendió de su propio error**, que es exactamente la definición de *harness engineering*: cada vez que algo falla, rediseñas para que no vuelva a pasar.

La anatomía, en cinco capas. El framework implementa los ocho componentes en cinco capas, y es **multi-plataforma por diseño** —una sola fuente que se construye para distintos entornos de agente—, lo que demuestra en la práctica que un harness bien diseñado no tiene por qué casarse con un solo proveedor:

Capa del framework	Qué hace	Componente(s) del harness
Agente	El cerebro conectado: prompt de sistema + recursos	Orquestación
Steering (siempre activo)	Reglas de seguridad, estándares y gobernanza en cada interacción	Guardrails
Skills (bajo demanda)	Flujos estructurados como herramientas invocables	Herramientas + orquestación
Hooks + guardia	Verificación automática: lint, escaneo de dependencias, validación	Verificación
Knowledge (grafos)	Memoria consultable entre iteraciones, *solo lectura* hacia el framework	Memoria + contexto

FIGURA 3

Componentes del harness frente a capas del framework 3HTP

HARNESS · COBERTURA DEL FRAMEWORK

Seis de los ocho componentes del harness tienen capa propia en el framework

Componente del harness	Agente	Steering	Skills	Hooks + guardia	Knowledge	Estado
Contexto	○	○	○	○	●	Capa propia
Herramientas (MCP, APIs)	○	○	●	○	○	Capa propia
Memoria y estado	○	○	○	○	●	Capa propia
Orquestación y bucle de control	●	○	●	○	○	Capa propia
Entorno de ejecución	○	○	○	○	○	Delegado a la plataforma
Guardrails y permisos	○	●	○	○	○	Capa propia
Verificación	○	○	○	●	○	Capa propia
Observabilidad y trazabilidad	○	○	○	○	○	Sin capa dedicada

● Cubierto por una capa del framework ○ Delegado o sin capa propia

Nota: la observabilidad aparece sin capa dedicada; pendiente de confirmar con el equipo del framework.
Fuente: elaboración propia a partir de la estructura de capas del framework 3HTP.

Nota. Elaboración propia a partir de la estructura de capas del framework.

Un ejemplo concreto de verificación en capas: la revisión de código del framework separa a *quien encuentra* un hallazgo de seguridad de *quien lo aprueba*. Cada hallazgo bloqueante pasa por un verificador independiente, con contexto fresco, cuyo único trabajo es *intentar refutarlo*. Ese patrón reduce el riesgo de aprobar hallazgos falsos por inercia. Es verificación como componente de primera clase del harness, no como paso opcional.

Honestidad de alcance. Este es el harness de trabajo de una consultora, no una plataforma de hiperescala; su valor como caso está en el *diseño*, no en un volumen que no vamos a inflar. Y

hay componentes que delega deliberadamente a la plataforma anfitriona (el aislamiento de ejecución, por ejemplo, lo provee el entorno donde corre). Un mapeo perfecto sin huecos sería sospechoso; este los tiene y los declara.

Conclusión

El modelo es una pieza extraordinaria, y sí, elegir uno capaz es la condición de entrada. Pero el modelo razona; **el harness es quien responde por lo que ese razonamiento hace en tus sistemas.** Es la capa que decide qué ve el agente, qué puede tocar, qué recuerda, dónde se ejecuta y cómo se le verifica. A medida que las tareas se alargan y los agentes se multiplican, esa capa deja de ser un detalle de implementación y pasa a ser la superficie primaria de control, seguridad y confiabilidad.

Antes de escalar tu stack de agentes, haz un ejercicio simple: mapea tus agentes contra los ocho componentes del harness. El que esté vacío es tu próximo incidente.

> **Diseñar y operar harnesses confiables —agnósticos de modelo, plataforma y nube— es nuestro oficio en 3HTP. Contáctenos para una sesión de arquitectura sin costo** y evaluemos juntos la madurez del harness de sus agentes, componente por componente, antes de llevarlos a producción.

Sobre el autor

Sebastian De La Vega es Cloud Engineer en 3HTP Cloud Services. Tiene la certificación AWS Certified Solutions Architect y formación fundamental en Anthropic. Lidera iniciativas de inteligencia artificial y diseñó el framework de desarrollo de 3HTP, el harness con el que la organización opera sus propios proyectos y que sirve de caso en este artículo.

Escribe para cerrar la brecha entre la velocidad con que los equipos adoptan agentes y las prácticas que deberían acompañarlos. Contacto: LinkedIn.

Referencias

Amazon Web Services. (s. f.). *AWS Well-Architected Agentic AI Lens*. https://docs.aws.amazon.com/wellarchitected/latest/agentic-ai-lens/agentic-ai-lens.html

Amazon Web Services. (s. f.). *Security and access controls: Amazon Bedrock AgentCore harness*. Documentación de AWS. https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/harness-security.html

Anthropic. (s. f.). *Agent SDK overview*. Claude Code Docs. https://code.claude.com/docs/en/agent-sdk/overview

Anthropic. (s. f.). *Claude Managed Agents overview*. Claude Platform Docs. https://platform.claude.com/docs/en/managed-agents/overview

Anthropic. (2025). *How we built our multi-agent research system*. Anthropic Engineering. https://www.anthropic.com/engineering/multi-agent-research-system

Anthropic. (2026). *Measuring AI agent autonomy in practice*. https://www.anthropic.com/news/measuring-agent-autonomy

Chase, H. (2025). *Agent frameworks, runtimes, and harnesses: Oh my!* LangChain Blog. https://www.langchain.com/blog/agent-frameworks-runtimes-and-harnesses-oh-my

Gartner. (2026). *Gartner identifies six steps to manage AI agent sprawl* [Comunicado de prensa]. https://www.gartner.com/en/newsroom/press-releases/2026-04-28-gartner-identifies-six-steps-to-manage-artificial-intelligence-agent-sprawl

GitHub. (s. f.). *Langflow has remote code execution in CSV Agent* (CVE-2026-27966; GHSA-3645-fxcv-hqr4) [Aviso de seguridad]. GitHub Advisory Database. https://github.com/advisories/GHSA-3645-fxcv-hqr4

Gu, S. (2026). *From model scaling to system scaling: Scaling the harness in agentic AI* (arXiv:2605.26112). arXiv. https://doi.org/10.48550/arXiv.2605.26112

Hugging Face. (s. f.). *Harness, scaffold, and the AI agent terms worth getting right*. Hugging Face Blog. https://huggingface.co/blog/agent-glossary

LangChain. (s. f.). *Deep Agents overview*. LangChain Docs. https://docs.langchain.com/oss/python/deepagents/overview

Lee, Y., et al. (2026). *Meta-Harness: End-to-end optimization of model harnesses* (arXiv:2603.28052). arXiv. https://doi.org/10.48550/arXiv.2603.28052

Lou, X., et al. (2026). *AutoHarness: Improving LLM agents by automatically synthesizing a code harness* (arXiv:2603.03329). arXiv. https://doi.org/10.48550/arXiv.2603.03329

METR. (2025). *Measuring AI ability to complete long software tasks*. https://metr.org/blog/2025-03-19-measuring-ai-ability-to-complete-long-tasks/

Microsoft. (s. f.). *Agent harness*. Microsoft Learn. https://learn.microsoft.com/en-us/agent-framework/concepts/harness

OWASP Foundation. (s. f.). *Artificial Intelligence Security Verification Standard (AISVS)*. https://owasp.org/www-project-artificial-intelligence-security-verification-standard-aisvs-docs/