Facsímil 05

Agentes y orquestación

Capítulo 07

SDKs de agentes: OpenAI, Anthropic, Google ADK y herramientas

686f6c61 · 2026

Capítulo 07: SDKs de agentes: OpenAI, Anthropic, Google ADK y herramientas

Cuando instalar un SDK parece arquitectura

Hay una tentación muy normal: ves un SDK de agentes, copias el ejemplo mínimo, consigues que el modelo llame una tool y sientes que ya tienes arquitectura. La demo funciona. El problema empieza cuando necesitas cambiar de modelo, añadir trazas, limitar tools, guardar sesiones, evaluar trayectorias, desplegar en otro entorno o explicar por qué el agente tomó una decisión.

El SDK es importante, pero no es el sistema entero. Un SDK te da una forma concreta de hablar con una plataforma. Tu producto necesita algo más estable: contratos internos, adaptadores, estado propio, evaluación, observabilidad, permisos, política de memoria y una forma de salir de un proveedor si mañana cambia el coste, la API o la capacidad.

En el capítulo 03 vimos qué es una tool bien diseñada. En el capítulo 04 vimos contexto, memoria y handoff. En el capítulo 06 rodeamos el agente con harness, límites y trazas. Ahora conectamos esas piezas con SDKs reales: OpenAI Agents SDK, Claude Agent SDK/API, Google ADK y el ecosistema de herramientas.

Qué no es elegir un SDK

Elegir un SDK no es elegir “el mejor modelo”. El modelo puede cambiar dentro del mismo SDK.

Tampoco es elegir “el proveedor para siempre”. Si acoplas tus tools, sesiones, errores y trazas al formato exacto de una plataforma, has tomado una decisión de arquitectura aunque nadie la haya escrito.

Y no es decidir que todo debe vivir dentro del SDK. Hay estado que debería vivir en tu base de datos, permisos que deberían vivir en tu sistema, métricas que deberían vivir en observabilidad y contratos de tools que deberían poder probarse sin llamar al modelo.

La pregunta profesional no es:

> “¿Qué SDK uso?”

La pregunta profesional es:

> “¿Qué parte de mi sistema quiero que resuelva el SDK, qué parte controlo yo y qué coste de cambio acepto?”

La definición útil

Para este capítulo, un SDK de agentes es:

> Una capa de desarrollo que facilita ejecutar bucles agentic: preparar contexto, llamar al modelo, exponer tools, gestionar sesiones, hacer handoffs, emitir eventos, aplicar guardrails y devolver resultados al producto.

El SDK puede ser muy ligero, como una librería de cliente que llama una API de mensajes. O puede ser bastante opinado, como un runtime que ya trae agentes, runners, tools, handoffs, sesiones y trazas.

La distinción clave es esta:

NIVEL	QUÉ TE DA	EJEMPLO
API de modelo	Endpoint para enviar mensajes, tools y recibir salida.	Anthropic Messages API, OpenAI Responses API.
SDK de cliente	Tipos, métodos, streaming, errores y autenticación.	`openai`, `anthropic`, `@anthropic-ai/sdk`.
SDK de agentes	Bucle agentic, tools, sesiones, handoffs, trazas.	OpenAI Agents SDK, Claude Agent SDK, Google ADK.
Framework de orquestación	Grafos, estados, workflows, persistencia, retries.	LangGraph, LlamaIndex Workflows, CrewAI.
Protocolo	Interoperabilidad entre tools o agentes.	MCP para tools/contexto, A2A para agentes.

La anatomía formal de una integración

Toda integración con un SDK, sea cual sea el proveedor, tiene las mismas piezas. Eso no es una ecuación de la literatura, es una lista de ingeniería, así que la presentamos como tabla:

PIEZA	QUÉ FIJA	EJEMPLO
Proveedor P	Plataforma.	OpenAI, Anthropic, Google ADK.
Modelo M	Familia configurada.	Modelo fuerte para revisión, barato para clasificación.
Agentes A	Roles definidos.	Coordinador, revisor APA, verificador de fuentes.
Tools T	Capacidades.	`validar_cita`, `buscar_fuente`, `normalizar_apa`.
Sesión S	Estado.	Historial, `run_state`, memoria, checkpoints.
Contexto K	Constructor de prompt.	Instrucciones, documentos, memoria, artefactos.
Handoffs H	Delegaciones.	Coordinador a especialista.
Guardrails G	Gates.	Validar JSON, limitar tools, exigir citas.
Evaluación E	Medición.	Dataset de casos, métricas, trayectoria esperada.
Traza τ	Observabilidad.	Eventos de modelo, tool, handoff, coste, latencia.
Política Ω	Operación.	Timeouts, retries, presupuesto, permisos, fallback.

Un SDK serio reduce código repetitivo, pero no elimina estas piezas. Si no las ves en el SDK, siguen existiendo en tu producto; si no las diseñas, aparecen como comportamiento implícito.

Para decidir cuánto te ata un SDK, una medida útil es la **portabilidad**: la fracción de tu sistema que controlas tú (contratos, trazas, tests y estado propios) frente al total, incluidas las dependencias atadas a un SDK concreto. Es la misma idea que las métricas de acoplamiento e inestabilidad de un componente: cuanto más dependes de lo específico, más caro es cambiar.[1]

$$\text{portabilidad} = \frac{N_{\text{propios}}}{N_{\text{propios}} + N_{\text{específicos del SDK}}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
N_{propios}	Piezas que controlas tú.	Tool schemas internos, trazas propias, tests, estado.
$N_{\text{específicos del SDK}}$	Piezas atadas a un SDK concreto.	Handoff solo en una librería, session store propietario.
portabilidad	Margen de cambio, de 0 a 1.	0,70 indica que gran parte no depende del proveedor.

En palabras: si todo depende del SDK, la portabilidad tiende a 0 y cambiar de proveedor será caro; si la lógica vive en contratos propios, la portabilidad sube y el SDK es reemplazable.

Fecha de corte del estado del arte

Fecha de corte: 10 de junio de 2026.
Fuentes consultadas ese día: documentación oficial de OpenAI Agents SDK para Python y JavaScript; documentación oficial de Anthropic para Claude Agent SDK, Messages API, tool use y MCP connector; documentación oficial de Google ADK sobre agentes, tools, sesiones, memoria y evaluación; especificación pública de MCP; especificación A2A; papers sobre uso de herramientas por LLMs.

Lo estable es el patrón: agente, tools, sesiones, handoff, trazas, evaluación, permisos y adaptadores. Lo cambiante son nombres de paquetes, modelos por defecto, encabezados beta, compatibilidad de tools, límites, precios, módulos de memoria, conectores y capacidades hospedadas.

La revisión del 10 de junio confirma que conviene enseñar SDKs como **capas de runtime**, no como recetas cerradas. OpenAI mantiene una separación clara entre agente, runner, handoffs, guardrails y tracing. Anthropic ya presenta Claude Agent SDK como biblioteca Python/TypeScript sobre el arnés de Claude Code, con permisos, hooks, sesiones, subagentes y herramientas incluidas. Google ADK y A2A empujan la interoperabilidad entre agentes, mientras que LangGraph sigue siendo una referencia práctica para ejecución durable, interrupciones humanas y persistencia. La regla de ingeniería se mantiene: diseña tu contrato interno antes del SDK y trata cada proveedor como un adaptador observable.

OpenAI Agents SDK: runtime opinado para agentes

OpenAI Agents SDK define `Agent` como el bloque principal: un LLM con instrucciones, tools y comportamiento opcional como handoffs, guardrails y salidas estructuradas.[2] La documentación oficial lo presenta como una forma de construir aplicaciones agentic en las que un modelo usa contexto, tools, handoffs, streaming y trazas.[3]

La idea importante: OpenAI te da una abstracción bastante completa. El `Runner` ejecuta el agente; las tools pueden ser funciones, tools hospedadas o agentes usados como tools; los handoffs permiten transferir una conversación a otro agente; las sesiones evitan reconstruir manualmente el historial; el tracing captura generaciones, tools, handoffs y guardrails.[4]

PIEZA EN OPENAI AGENTS SDK	QUÉ SIGNIFICA PARA INGENIERÍA
`Agent`	Unidad con instrucciones, modelo, tools, handoffs y configuración.
`Runner` / `run`	Runtime que ejecuta el bucle del agente.
Function tools	Funciones propias expuestas como tools con contrato.
Hosted tools	Tools gestionadas por OpenAI, como búsqueda, file search o code interpreter, según SDK y entorno.[5]
Agents as tools	Un agente especializado se expone como una tool del agente coordinador.
Handoffs	Un agente transfiere la conversación a otro especialista; el handoff aparece como tool para el modelo.[6]
Sessions	Memoria de conversación gestionada por una implementación de sesión.[7]
MCP	Integración con servidores MCP, incluyendo nombres prefijados por servidor para evitar colisiones.[8]

Cuándo encaja bien:

ENCAJA CUANDO...	CUIDADO CON...
Quieres un runtime de agente con trazas y handoffs ya pensados.	No ocultar reglas de negocio dentro de callbacks imposibles de probar.
Trabajas en Python o TypeScript y quieres moverte rápido.	No depender de una feature hospedada si necesitas portabilidad.
Necesitas usar tools de OpenAI y flujos con varios agentes.	Separar tool contract interno del decorador específico del SDK.
Quieres observar ejecuciones desde el principio.	Normalizar trazas si comparas con otros proveedores.

Anthropic: Messages API, Claude Agent SDK y Claude Code

Anthropic tiene dos niveles que conviene no mezclar. El primer nivel es la Messages API: una API de mensajes donde tú envías historial, system prompt, tools y recibes bloques de contenido. La documentación dice explícitamente que la Messages API es stateless: debes enviar el historial conversacional completo que quieras que el modelo vea.[9]

El segundo nivel es el Claude Agent SDK. La documentación actual indica que el Claude Code SDK fue renombrado a Claude Agent SDK, disponible para TypeScript y Python, y construido sobre el arnés de agentes que impulsa Claude Code.[10] En la quickstart, el punto central es `query`: la entrada que crea el bucle agentic y devuelve un iterador asíncrono para observar mensajes mientras Claude trabaja.[11]

PIEZA ANTHROPIC	QUÉ APORTA	CÓMO LEERLA
Messages API	Control bajo nivel de mensajes, tools y streaming.	Tú gestionas historial, estado, reintentos y ciclo de tools.
Tool use	Claude pide `tool_use`; tu aplicación ejecuta y devuelve `tool_result`.[12]	Muy transparente para entender el protocolo.
Claude Agent SDK	Runtime alto nivel para construir agentes con TypeScript o Python.	Útil si quieres reutilizar el arnés de Claude Code en tu producto.
Claude Code	Herramienta de desarrollo agentic con memoria, subagentes, hooks y MCP.	Buen ejemplo de harness real, aunque no todo aplica a cualquier producto.
MCP connector	Permite conectar servidores MCP remotos desde la Messages API; en la versión consultada usa beta header `mcp-client-2025-11-20` y soporta tools, con limitaciones claras.[13]	Muy útil para tools remotas, pero debes leer autenticación, retención y compatibilidad.

Anthropic brilla cuando quieres ver el protocolo con claridad. La Messages API obliga a entender tool use, historial y control externo. El Claude Agent SDK te da un arnés más completo. Claude Code enseña patrones prácticos: `CLAUDE.md`, subagentes, hooks, skills, permisos de tools y configuración por proyecto.

Cuándo encaja bien:

ENCAJA CUANDO...	CUIDADO CON...
Quieres control explícito de la conversación y de las tools.	La API de mensajes no guarda estado por ti.
Quieres construir agentes sobre el arnés de Claude Code.	El SDK de agente tiene supuestos fuertes sobre entorno y ejecución.
Trabajas con código, repositorios o workflows de desarrollo.	No confundas una herramienta de desarrollo con una arquitectura general de producto.
Quieres usar MCP remoto desde la API.	Verifica transporte, autenticación, retención y tools permitidas.

Anatomía del SDK de Anthropic

Cuando alguien dice “el SDK de Anthropic” puede estar hablando de tres cosas distintas. Conviene separarlas antes de decidir arquitectura:

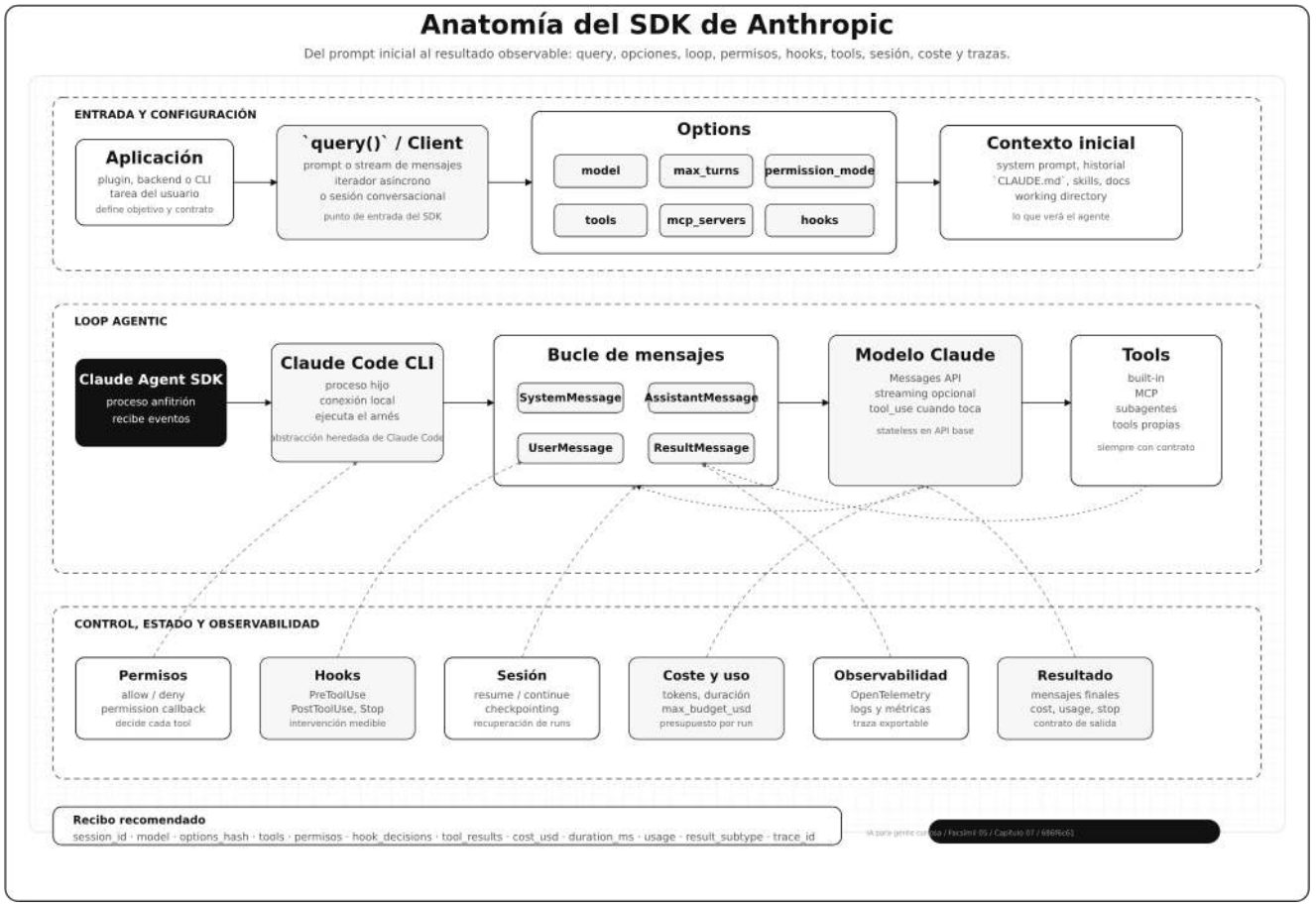
SUPERFICIE	QUÉ CONTROLAS TÚ	QUÉ RESUELVE ANTHROPIC	CUÁNDO USARLA
Messages API	Historial, tools, ciclo de ejecución, estado, streaming y validación.	El modelo, el protocolo de mensajes y los bloques `tool_use` / `tool_result`.	Cuando quieres un loop propio y máximo control.
SDK cliente	Tipos, cliente HTTP, autenticación, streaming y errores.	Acceso cómodo a la API desde Python, TypeScript u otro lenguaje soportado.	Cuando no necesitas arnés agentic completo.
Claude Agent SDK	Bucle agentic, interacción con Claude Code, tools, permisos, hooks, sesiones, coste y observabilidad.	Un runtime de agente ejecutado desde tu proceso, con eventos observables.	Cuando quieres construir sobre el arnés de Claude Code.
Claude Code	Proyecto local, `CLAUDE.md`, subagentes, comandos, skills, plugins, hooks y herramientas de desarrollo.	Un entorno agentic de trabajo sobre archivos, terminal y repositorios.	Cuando el dominio es ingeniería de software o workflows sobre workspace.

La Messages API es stateless: si quieres que Claude vea conversación anterior, debes enviar el historial que toca.[14] El Claude Agent SDK, en cambio, construye un agente sobre el arnés de Claude Code; la quickstart muestra `query()` como punto de entrada y devuelve mensajes de manera asíncrona mientras el agente trabaja.[15]

La anatomía del SDK de Anthropic son estas piezas (una lista de ingeniería, no una ecuación de la literatura):

SÍMBOLO	PIEZA	QUÉ SIGNIFICA EN INGENIERÍA
Q	`query` o `ClaudeSDKClient`	Entrada de trabajo y canal para recibir eventos.
O	Opciones	Modelo, directorio de trabajo, tools, presupuesto, permisos, MCP, hooks y sesiones.
C	Contexto	Prompt, historial, `CLAUDE.md`, subagentes, skills, documentos y estado del proyecto.
L	Loop agentic	Turnos de modelo, posibles tools, resultados de tools y salida final.
T	Tools	Built-in tools, tools MCP, tools propias y subagentes como capacidad especializada.
P	Permisos	Modos de permiso, allow/deny lists, callbacks y hooks antes de ejecutar tools.
H	Hooks	Puntos de intervención antes/después de tool, parada, subagente o notificación.
S	Sesión	Continuación, reanudación, checkpointing y estado conversacional.
R	Resultado	Mensajes, stream, `ResultMessage`, coste, duración, uso y razón de finalización.
Ω	Operación	Logs, OpenTelemetry, métricas, fallback, límites y pruebas de regresión.

Anatomía visual



La figura deja una idea importante: el Claude Agent SDK no es solo una llamada HTTP. Tu aplicación llama al SDK, el SDK configura una ejecución del arnés de Claude Code, ese arnés conversa con Claude, puede pedir tools, pasa por permisos y hooks, y al final devuelve mensajes, uso, coste, duración y estado de cierre. Si usas la Messages API directamente, muchas de esas cajas siguen existiendo, pero las implementas tú.

El bucle paso a paso

El bucle del Agent SDK se entiende mejor si lo leemos como una secuencia observable:

PASO	QUÉ OCURRE	QUÉ DEBERÍAS REGISTRAR
1. Entrada	Tu app llama `query()` o abre un `ClaudeSDKClient` con prompt y opciones.	`run_id`, `session_id`, versión de agente y hash de opciones.
2. Inicialización	El SDK prepara el proceso, el contexto, el directorio de trabajo y la configuración.	Directorio, modelo, tools permitidas, MCP, hooks y presupuesto.
3. Mensaje inicial	Llega un `SystemMessage` con metadatos de sesión y estado inicial.	Session id real y metadatos devueltos por el runtime.
4. Turno del modelo	Claude genera texto, decide seguir, o solicita una tool.	Tokens, latencia, bloques de contenido y motivo de parada si aplica.
5. Decisión de tool	El runtime comprueba permisos, listas, callbacks y hooks.	Nombre de tool, input, decisión, motivo y quién la autorizó.
6. Ejecución	Se ejecuta una tool built-in, MCP, propia o de subagente.	Resultado, error controlado, duración y efecto declarado.
7. Resultado de tool	El output vuelve al loop como información para Claude.	Tamaño del resultado y si se recortó o resumió.
8. Repetición	El loop continúa hasta salida final, límite, error o parada.	Número de turnos, tools usadas y coste acumulado.
9. Resultado	Llega un `ResultMessage` con subtipo, coste, duración y uso.	Salida validada, `usage`, `total_cost_usd`, `duration_ms` y estado final.

La documentación del Agent SDK describe mensajes como `SystemMessage`, `AssistantMessage`, `UserMessage` y `ResultMessage`, y también subtipos de resultado como éxito, error y límite de turnos.[16]

Permisos: la parte que no se debe improvisar

El SDK permite combinar `allowed_tools`, `disallowed_tools`, modos de permiso, callbacks y hooks. No son opciones sueltas: son una **política de ejecución** que aplica la mediación completa, el principio de comprobar cada acceso antes de actuar.[17] Cada tool call $D(t, a, c)$ atraviesa, en orden, una cadena de comprobaciones (hook previo, denegación, modo de permiso, lista de permitidas y callback), y solo se ejecuta si todas la autorizan:

PIEZA	QUÉ SIGNIFICA	QUÉ DECIDIRÍA EN UN PROYECTO SERIO
t	Tool solicitada.	Nombre estable y versión de schema.
a	Argumentos.	Validación antes de ejecutar.
c	Contexto de ejecución.	Usuario, tenant, directorio, sesión y presupuesto.
H_{pre}	Hook antes de tool.	Puede bloquear, modificar o registrar la solicitud.
`Deny(t)`	Lista o regla que no permite una tool.	Siempre gana sobre una allowlist.
`Mode(t)`	Modo de permisos de la ejecución.	Modo lectura, aceptar ediciones, omitir permisos o modo plan.
`Allow(t)`	Tools explícitamente disponibles.	Útil, pero no suficiente como auditoría.
`Callback(t,a,c)`	Función propia de autorización.	Donde conectas reglas de negocio.

La documentación oficial de permisos explica que hay varias capas: herramientas permitidas y no permitidas, modos de permiso y callbacks; también indica que las decisiones pueden integrarse con hooks previos a tool.[18] El matiz de ingeniería: una allowlist no sustituye a una política. La allowlist dice “esta tool existe para esta ejecución”; la política decide “esta llamada concreta, con estos argumentos, en este contexto, puede hacerse”.

Hooks: instrumentación y control

Los hooks son puntos de intervención del arnés. Sirven para observar, validar, modificar contexto o detener una operación antes de que ocurra. En una integración profesional no los usaría para esconder lógica de dominio, sino para conectar el agente con el sistema operativo del producto:

HOOK O MOMENTO	USO SANO	SEÑAL DE MALA ARQUITECTURA
Antes de tool	Validar argumentos, registrar intención, aplicar política.	Reescribir medio prompt porque no hay contrato claro.
Después de tool	Guardar resultado, medir latencia, normalizar errores.	Parsear salidas libres imposibles de testear.
Al parar	Emitir evento final, cerrar spans, persistir resumen.	Depender de logs manuales para saber qué pasó.
En subagente	Medir delegación y contexto entregado.	Delegar sin saber qué recibió el especialista.

Anthropic documenta hooks para personalizar el comportamiento del agente en momentos como tool use, parada, notificaciones y subagentes.[19] Si el capítulo 06 hablaba de harness, aquí se ve aplicado: el hook es un punto de control.

Mensajes, streaming y resultado

En Messages API, el streaming permite recibir eventos parciales: arranque de mensaje, bloques de contenido, deltas, parada de bloque y parada de mensaje.[20] En Agent SDK, el desarrollador ve mensajes de más alto nivel del loop. Esa diferencia importa:

SI USAS...	LO QUE VES	LO QUE TE TOCA CONTROLAR
Messages API sin streaming	Respuesta completa al final.	Historial, tools, reintentos y estado.
Messages API con streaming	Eventos de texto y bloques parciales.	Render incremental, cancelación, errores parciales.
Claude Agent SDK con `query()`	Secuencia de mensajes del agente.	Consumir eventos, persistir trazas y validar resultado.
`ClaudeSDKClient`	Conversación más interactiva.	Ciclo de vida de sesión y envío de nuevas entradas.

Una salida “correcta” no debería ser solo texto. Para un producto real pediría:

CAMPO	POR QUÉ
`result_subtype`	Distingue éxito, error o límite alcanzado.
`usage`	Permite comparar coste entre versiones.
`total_cost_usd`	Convierte tokens y tools en presupuesto entendible.
`duration_ms`	Ayuda a detectar latencia por modelo, tool o red.
`trace_id`	Une logs, spans y eventos del producto.
`final_schema_valid`	Separa “parece correcto” de “cumple contrato”.

Sesión, checkpointing y continuidad

La sesión responde a una pregunta: “¿cómo continúa una ejecución o conversación sin reconstruir todo a mano?”. El checkpointing responde a otra: “¿puedo guardar y reanudar desde un punto fiable?”. No son la misma cosa que memoria semántica.

CONCEPTO	QUÉ CONSERVA	RIESGO SI SE CONFUNDE
Historial de conversación	Turnos y mensajes.	Creer que todo historial es conocimiento útil.
`session_id`	Identidad de una sesión.	Mezclar sesiones de usuarios o tareas distintas.
Checkpoint	Punto recuperable de una ejecución.	No poder depurar una run larga fallida.
Memoria externa	Hechos reutilizables entre sesiones.	Meter recuerdos irrelevantes en cada prompt.
`CLAUDE.md`	Instrucciones persistentes del proyecto.	Convertir normas locales en verdad universal.

Anthropic documenta continuidad de sesión y checkpointing en el Agent SDK, útiles cuando una ejecución necesita reanudarse o conservar estado operacional.[21] La regla para nuestro facsímil: sesión es continuidad de trabajo; memoria es conocimiento recuperable; contexto es lo que entra en una llamada concreta.

Coste y observabilidad

El Agent SDK expone coste y uso en los resultados, y documenta `max_budget_usd` como control de presupuesto por ejecución.[22] También documenta observabilidad con OpenTelemetry para exportar métricas, logs y trazas.[23]

Para un ingeniero, esto cambia la forma de evaluar:

MÉTRICA	QUÉ MIDE	UMBRAL ÚTIL
`turn_count`	Cuántas vueltas necesita el agente.	Si crece, quizá falta tool o prompt más claro.
`tool_call_count`	Cuántas herramientas usa.	Si se dispara, hay mala planificación o mala recuperación.
`permission_denied_count`	Cuántas acciones fueron rechazadas.	Si es alto, el agente no entiende límites.
`total_cost_usd`	Coste total de la run.	Debe mirarse por tarea aceptada, no por demo.
`duration_ms`	Tiempo real de la ejecución.	Separa latencia de modelo, tools y cola.
`schema_valid_rate`	Porcentaje de salidas válidas.	Si baja, falta contrato o reparación controlada.
`resume_success_rate`	Capacidad de recuperar sesiones/checkpoints.	Clave en runs largas.

Cómo lo integraría en nuestro contrato portable

CONTRATO DEL FACSÍMIL	ANTHROPIC LO RESUELVE CON...	QUÉ DEJARÍA FUERA DEL SDK
`AgentSpec`	Prompt, opciones y configuración del agente.	Identidad del agente, versión y criterios de cierre.
`ToolSpec`	Tools de Messages API, tools del Agent SDK o MCP.	Schema canónico, efecto e idempotencia.
`PermissionPolicy`	`allowed_tools`, `disallowed_tools`, modo, callback y hooks.	Reglas de negocio y auditoría centralizada.
`SessionStore`	Session id, continuidad y checkpointing.	Propietario real del dato y reglas de borrado.
`TraceEvent`	Mensajes del loop, OTel, logs y resultado.	Formato común entre proveedores.
`EvalDataset`	Datos propios de evaluación.	Golden traces, métricas y umbrales de aceptación.
`CostEnvelope`	`max_budget_usd`, usage y coste.	Presupuesto por usuario, tenant o tarea.

Lo que me gusta de Anthropic para enseñar ingeniería es que obliga a mirar el loop. La Messages API te muestra el protocolo desnudo. El Agent SDK te da un arnés más completo, pero sigue siendo observable si consumes mensajes, coste, permisos y hooks. La trampa sería lo de siempre: usarlo como caja negra y llamar arquitectura a una demo.

Google ADK: framework de agentes, sesiones, memoria y evaluación

Google ADK se presenta como un kit de desarrollo para construir y desplegar agentes. Su documentación define un agente o `LlmAgent` como una unidad autocontenida que puede perseguir objetivos, interactuar con usuarios, usar tools y coordinarse con otros agentes.[24]

La documentación técnica destaca varias piezas relevantes: memoria para recordar información entre sesiones, evaluación integrada para crear datasets multi-turn y ejecutar evaluaciones localmente, y soporte amplio de LLMs mediante interfaces como `BaseLlm`, aunque esté optimizado para Gemini.[25]

PIEZA GOOGLE ADK	QUÉ APORTA	CÓMO LEERLA
`Agent` / `LlmAgent`	Unidad de ejecución con modelo, instrucciones y tools.	Similar en idea a otros SDKs, con sabor Google/Gemini.
`Runner`	Ejecuta el agente con servicios de sesión y memoria.	Separa definición de agente y ejecución.
Session service	Historial, eventos y estado de una conversación.	Estado corto, no memoria permanente.
Memory service	Memoria a largo plazo; puede usarse con tools como `load_memory` o `PreloadMemoryTool`.[26]	Requiere decidir cuándo pasar sesiones a memoria.
Tools	Funciones, herramientas de Google Cloud, MCP Toolbox, conectores y herramientas de terceros.[27]	Muy orientado a ecosistema enterprise y Google Cloud.
Evaluation	Evalúa trayectoria y respuesta con datasets y criterios.[28]	Valioso para no quedarse en demos.
A2A	Integración con Agent2Agent para comunicación entre agentes.[29]	Lo veremos con más detalle en el capítulo 09.

Google ADK encaja especialmente bien si quieres un framework completo con agentes, servicios de sesión/memoria, evaluación y despliegue cercano al ecosistema Google. También es interesante para enseñar arquitectura porque fuerza a separar agente, runner, sesión, memoria y evaluación.

MCP y A2A: no son lo mismo que un SDK

MCP y A2A aparecen mucho al hablar de SDKs, pero cumplen otra función.

MCP, Model Context Protocol, estandariza cómo una aplicación ofrece contexto y herramientas a modelos o agentes. Es una frontera de herramientas: servidores, tools, recursos, prompts, transportes y autorización.[30]

A2A, Agent2Agent Protocol, busca interoperabilidad entre sistemas agentic independientes. Es una frontera de agentes: descubrir capacidades, enviar tareas, mantener contexto de conversación y coordinar trabajo entre sistemas.[31]

TECNOLOGÍA	FRONTERA PRINCIPAL	PREGUNTA QUE RESPONDE
SDK de modelo	Aplicación ↔ modelo	¿Cómo llamo al modelo?
SDK de agentes	Aplicación ↔ runtime agentic	¿Cómo ejecuto bucles con tools, estado y trazas?
MCP	Agente ↔ tools/contexto	¿Cómo expongo capacidades externas de forma estándar?
A2A	Agente ↔ agente	¿Cómo conversa un agente con otro sistema agentic?

Regla práctica: no uses MCP para sustituir diseño de tools; úsalo para empaquetar y exponer tools. No uses A2A para esconder falta de arquitectura interna; úsalo cuando realmente hay sistemas agentic independientes que deben coordinarse.

Mercado y criterio de elección

Además de OpenAI, Anthropic y Google, existen LangGraph, LlamaIndex, CrewAI, AutoGen, Haystack, Semantic Kernel, Vercel AI SDK, OpenCode, Codex CLI, Claude Code, Cursor y otros entornos. La lista cambia rápido. El criterio no debería ser popularidad, sino ajuste al tipo de sistema.

NECESITAS...	PRIORIZA...	PREGUNTA INCÓMODA
Runtime agentic completo con trazas	OpenAI Agents SDK, Google ADK, LangGraph.	¿Puedo exportar o normalizar las trazas?
Protocolo claro de tool use	Anthropic Messages API, OpenAI Responses API.	¿Dónde vive el estado entre turnos?
Agente de código con entorno de trabajo	Claude Code, Codex CLI, OpenCode, Cursor.	¿Qué permisos y rutas puede tocar?
Integración enterprise con Google Cloud	Google ADK + Vertex/Google Cloud tools.	¿Dependo de servicios concretos del cloud?
Tools reutilizables entre clientes	MCP.	¿Tengo allowlist, auth y nombres únicos?
Agentes entre organizaciones o sistemas	A2A.	¿Necesito interoperabilidad o solo una tool?
Máxima portabilidad	Adapter propio + contratos internos.	¿Qué pierdo si no uso features nativas?

La respuesta madura casi nunca es “todo abstracto” ni “todo nativo”. Lo normal es una mezcla:

1. Contrato interno propio para tools, trazas, sesiones y resultados.
2. Adapter fino por proveedor.
3. Uso consciente de features nativas cuando aportan mucho.
4. Evals que comparan comportamiento, no solo compilación.

El patrón que mantiene alta la portabilidad es viejo y conocido: el **adaptador**.[32] Defines tu propia interfaz (`AgentSpec` , `ToolSpec` , `TraceEvent`) y escribes un adaptador fino por cada SDK. Tu dominio habla con tu interfaz; el adaptador traduce a OpenAI, Anthropic o Google ADK. Así, cambiar de proveedor es reescribir un adaptador, no el producto.

Qué le faltaba al capítulo

Antes de dibujar arquitectura, hay cuatro preguntas que un capítulo universitario sobre SDKs no debería esquivar:

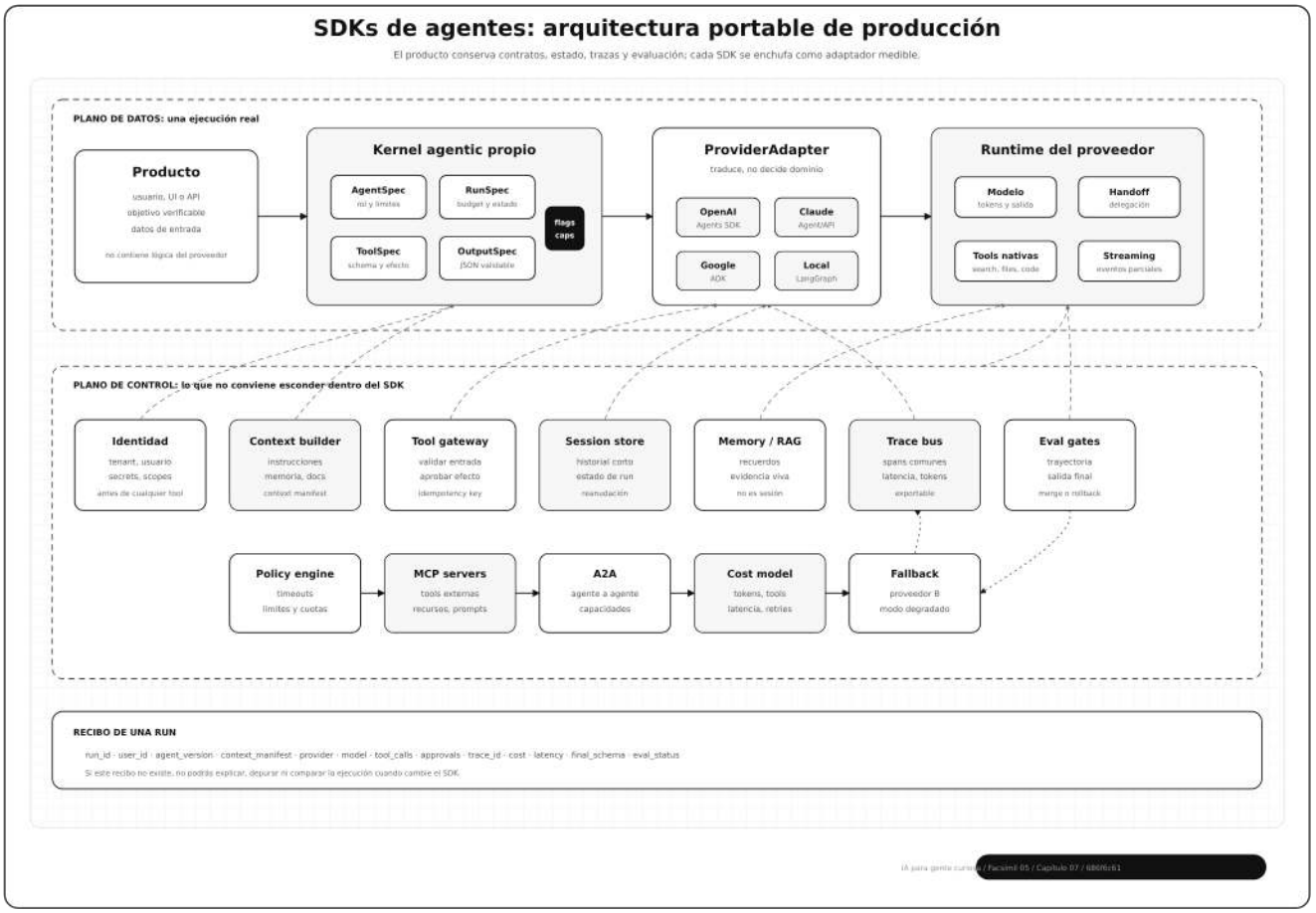
FALTA HABITUAL	POR QUÉ IMPORTA	QUÉ DEBERÍAMOS PRODUCIR
Criterio de adopción	Un SDK puede acelerar o encerrar el diseño.	Matriz: API directa, SDK de agente, framework de grafos, MCP o A2A.
Contrato de ejecución	Una demo no describe retries, coste, estado ni salida.	`RunSpec` con presupuesto, límites, session id, trace id y esquema final.
Plano de control	El modelo no debe decidir credenciales, permisos, memoria y auditoría.	Tool gateway, policy engine, secretos, trazas, evals y fallback fuera del modelo.
Plan de migración	Cambiar de proveedor sin plan suele implicar reescribir producto.	Adaptadores finos y tests que comparan comportamiento entre proveedores.

La decisión se puede convertir en una regla práctica:

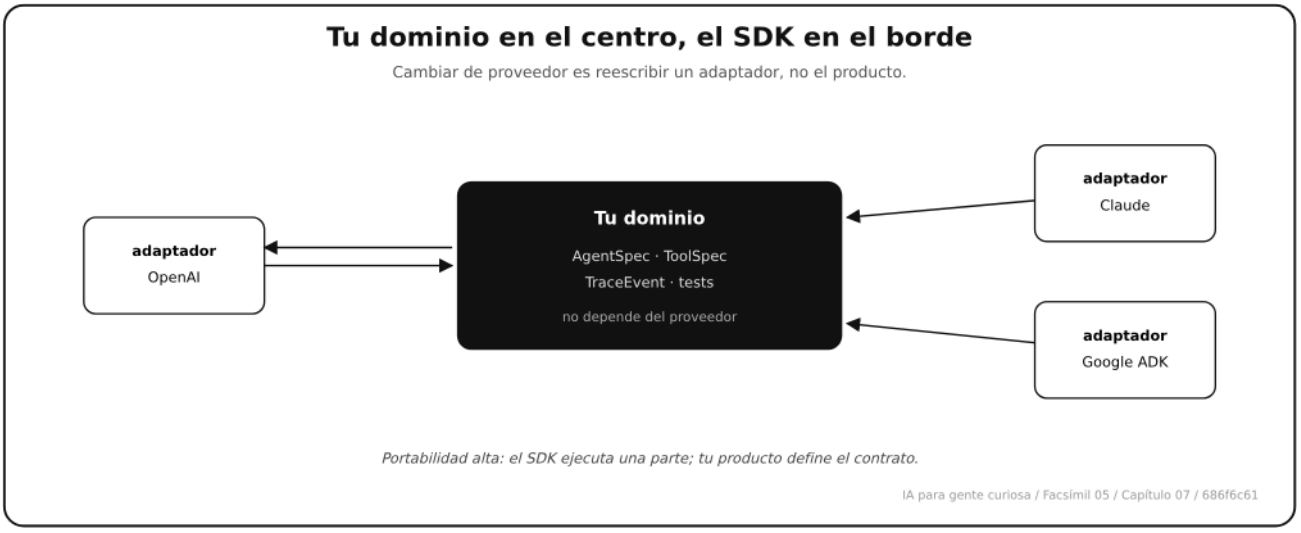
SI EL SISTEMA...	EMPIEZA POR...	NO EMPIECES POR...
Solo necesita una respuesta estructurada y pocas tools.	API de modelo + loop propio pequeño.	Framework pesado de agentes.
Necesita handoffs, tools, sesiones y trazas desde el día uno.	SDK de agentes.	Cliente HTTP escrito a mano sin observabilidad.
Tiene workflows largos, ramas y estado persistente.	LangGraph, Google ADK, LlamaIndex Workflows o framework equivalente.	Un único agente con prompt gigante.
Quiere reutilizar tools entre clientes distintos.	MCP.	Copiar la misma tool en cada proveedor.
Necesita que sistemas agentic independientes cooperen.	A2A o protocolo equivalente.	Encadenar agentes como si fueran simples funciones.
Tiene exigencia fuerte de portabilidad.	Contrato interno + adapters.	Usar tipos del SDK como modelo de dominio.

Otra forma de verlo: el SDK se elige después de saber qué parte quieres delegar. Si el SDK decide tu estado, tus tools, tus trazas y tu evaluación, ya no es una librería: es el esqueleto del producto.

Arquitectura visual de una integración portable



La arquitectura separa tres planos. El plano de datos es la ejecución visible: producto, contratos, adapter y runtime. El plano de control contiene lo que no debería quedar escondido dentro del SDK: identidad, contexto, tools, sesión, memoria, trazas, evaluación, coste y fallback. El recibo final de la run es la prueba de madurez: si no puedes reconstruir qué ocurrió, no puedes comparar proveedores ni depurar un cambio.



Reglas de integración que pondría en un proyecto real

Estas reglas son deliberadamente concretas.

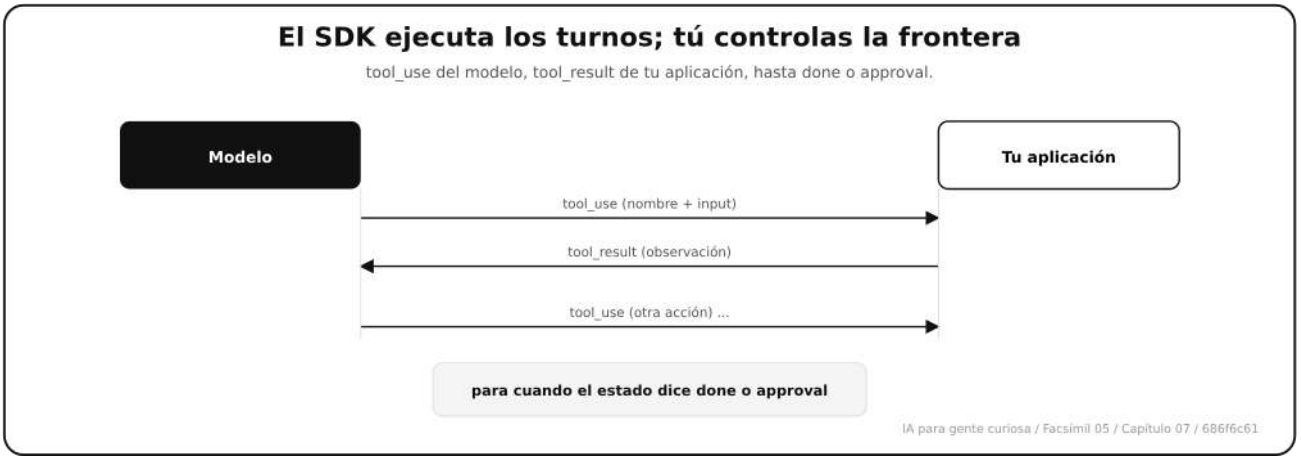
REGLA	POR QUÉ IMPORTA	SEÑAL DE QUE LO HACES BIEN
Define `AgentSpec` propio antes de instanciar el SDK.	Evita que tu arquitectura sea el ejemplo de la documentación.	Puedes imprimir un manifiesto de agente sin proveedor.
Define `ToolSpec` propio.	La tool pertenece a tu dominio, no al SDK.	Puedes ejecutar tests de tools sin modelo.
Usa capability flags.	Cada proveedor soporta piezas distintas.	`supports_handoffs`, `supports_mcp`, `supports_native_tracing`.
Guarda trazas normalizadas.	Comparar SDKs exige lenguaje común.	Todos emiten `model_call`, `tool_call`, `handoff`, `final_output`.
Separa sesión de memoria.	La sesión no siempre es recuerdo duradero.	Hay `session_store` y `memory_store` distintos.
Versiona instrucciones y schemas.	Los cambios de prompts y tools son cambios de software.	Cada run guarda `agent_version` y `tool_version`.
Haz retry solo con idempotencia.	Repetir una tool puede duplicar efectos.	Cada tool con efecto tiene `idempotency_key`.
No expongas tools genéricas.	Una tool demasiado amplia rompe el contrato.	Tools pequeñas, con precondiciones y salida limitada.
Evalúa trayectoria, no solo respuesta.	Un agente puede acertar mal.	El test mira steps, tools, coste y estado final.
Escribe plan de salida del proveedor.	Reduce dependencia accidental.	Sabes qué se perdería al migrar y cuánto costaría.

Parámetros que debes mapear entre SDKs

Aunque los nombres cambien, casi todos los sistemas agentic tienen estas piezas.

CONCEPTO	OPENAI	ANTHROPIC	GOOGLE ADK	CONTRATO PROPIO RECOMENDADO
Instrucciones	`instructions` del `Agent`.	System prompt, Agent SDK prompt o configuración.	`instruction` del agente.	`agent.instructions_version`.
Modelo	`model` o settings.	`model` en Messages/SDK.	`model` en `Agent`.	`model_profile`.
Tools	Function tools, hosted tools, MCP.	`tools`, MCP connector, Agent SDK tools.	Function tools, built-ins, MCP/Google tools.	`ToolSpec[]`.
Handoff	`handoffs`.	Subagentes/Agent SDK o routing propio.	Multi-agent/A2A/workflows.	`DelegationPolicy`.
Sesión	`Session` implementation.	Historial enviado o Agent SDK runtime.	`SessionService`.	`SessionStore`.
Memoria	Sessions, custom memory, external store.	`CLAUDE.md`, memory tools, store externo.	`MemoryService`.	`MemoryStore`.
Trazas	Tracing del Agents SDK.	Eventos SDK, logs, traces propios.	Evaluation/logging/Cloud observability.	`TraceEvent`.
Salida estructurada	Output types/schema.	Structured outputs o tool/result contract.	Schema/response contract.	`OutputSchema`.
Evaluación	Evals/trace grading/propias.	Tests/evals propias.	ADK evaluation.	`EvalDataset + rubric`.
MCP	SDK MCP integration.	MCP connector y helpers.	MCP tools / toolbox.	`ToolTransport`.

Esta tabla es más importante que el tutorial de instalación. Si no sabes mapear estos conceptos, no estás integrando un SDK: estás pegando código.



Ingeniería de producción: la run como contrato

Una integración agentic no debería empezar en `client.responses.create`, `query(...)` o `Runner.run(...)`. Debería empezar con una descripción completa de la ejecución, la **run como contrato**: un registro con todos los campos que hacen la ejecución reproducible y auditable.

SÍMBOLO	QUÉ REPRESENTA	PREGUNTA DE INGENIERÍA
u	Usuario, tenant o proceso que inicia la run.	¿Con qué permisos actúa?
x	Entrada original.	¿Se conserva sin mezclarla con memoria o sistema?
c	Context manifest.	¿Qué instrucciones, documentos y recuerdos entraron?
a	Agente o grafo elegido.	¿Qué versión exacta del agente se ejecutó?
p	Provider adapter.	¿Qué SDK, modelo y parámetros concretos se usaron?
b	Presupuesto.	¿Cuántos pasos, tokens, tools, coste y tiempo se permiten?
τ	Traza.	¿Puedo reconstruir cada llamada, tool y handoff?
o	Salida validada.	¿Cumple el esquema o solo parece correcta?
e	Evaluación posterior.	¿Pasó gates de trayectoria, calidad y coste?

El coste tampoco es una etiqueta genérica. El coste de una run es la suma de sus partes, una identidad de coste (la misma lógica del coste total de propiedad, no un teorema):[33]

$$C_R = C_{\text{input}} + C_{\text{output}} + \sum_i C_{\text{tool}_i} + C_{\text{retries}} + C_{\text{observabilidad}} + C_{\text{latencia}}$$

TÉRMINO	QUÉ MIDE	EJEMPLO DE DECISIÓN
C_{input}	Tokens de instrucciones, historial, RAG y memoria.	Compactar sesión o recortar contexto.
C_{output}	Tokens generados y razonamiento visible/no visible según proveedor.	Exigir salida corta y estructurada.
$\sum_i C_{\text{tool}_i}$	APIs, búsquedas, ejecución de código o consultas externas.	Cachear herramientas de lectura.
C_{retries}	Reintentos por timeout, JSON inválido o proveedor no disponible.	Reintentar solo operaciones idempotentes.
$C_{\text{observabilidad}}$	Trazas, logs, almacenamiento y redacción de datos sensibles.	Muestrear trazas en producción sin perder incidentes.
C_{latencia}	Tiempo de espera del usuario y ocupación de workers.	Streaming, colas o modo asíncrono.

Y hay fallos que conviene diseñar antes de verlos en producción:

CASO	QUÉ SUELE PASAR	DISEÑO QUE LO EVITA
Streaming parcial	El usuario ve media respuesta y luego falla una tool.	Eventos tipados, estado `partial`, reanudación y mensaje final coherente.
JSON inválido	El modelo devuelve algo cercano al esquema, pero no parseable.	Validador estricto, reparación limitada y error observable.
Tool lenta	La ejecución agota timeout y el agente queda esperando.	Timeout por tool, fallback y respuesta con lo que sí se sabe.
Tool repetida	Un retry duplica una acción externa.	`idempotency_key`, efecto declarado y confirmación en tools de escritura.
Contexto excesivo	El modelo recibe demasiado ruido.	Context manifest, ranking, compaction y evaluación de recuperación.
Cambio del SDK	Un nombre, evento o tipo deja de encajar.	Adapter con tests de contrato y versionado de provider.
Diferencia entre proveedores	Un mismo prompt no produce la misma trayectoria.	Eval de trayectoria por proveedor, no solo golden answer final.

La documentación actual de OpenAI Agents SDK ya separa agentes, tools, handoffs, guardrails, output types, lifecycle hooks y tracing; además el tracing captura generaciones, tools, handoffs, guardrails y spans propios.[34][35] Anthropic distingue claramente la Messages API stateless, donde debes reenviar el historial que quieras que el modelo vea, del Claude Agent SDK, que ejecuta el bucle agentic en tu proceso y aporta tools, contexto y observabilidad propias de Claude Code.[36][37] Google ADK, por su parte, explicita agentes, tools, callbacks, sesiones, memoria, artefactos, runners, evaluación y despliegue; en evaluación distingue trayectoria y respuesta final.[38][39]

La conclusión técnica es sencilla: si cada proveedor ya piensa en runtime, eventos y evaluación, nuestro facsímil no puede quedarse en “instala este SDK”. Debe enseñar a separar contrato, adapter y operación.

Caso concreto: un plugin de revisión académica con tres agentes

Imaginemos un plugin sencillo para este facsímil. Queremos revisar un párrafo antes de publicarlo. El sistema tiene tres especialistas:

1. Un revisor de normas RAE: detecta problemas de ortografía, mayúsculas, tildes y estilo.
2. Un revisor APA: revisa si las citas y referencias siguen un formato consistente.

3. Un verificador de fuentes: abre URLs o documentos y comprueba si la afirmación citada está soportada.

El coordinador no debe “hacerlo todo”. Debe decidir qué especialista usar, reunir evidencias, devolver un informe y decir qué no puede verificar.

PIEZA	CONTRATO INTERNO	EN OPENAI	EN CLAUDE	EN GOOGLE ADK
Coordinador	`agent: academic_reviewer`	`Agent` con agentes como tools o handoffs.	`query` con subagentes o loop de tools.	`LlmAgent` que coordina subagentes.
RAE	Tool/agent de revisión lingüística.	Agent as tool.	Subagent o tool `revisar_rae`.	`AgentTool` o subagente.
APA	Tool/agent de citas.	Agent as tool.	Subagent o tool `revisar_apa`.	`AgentTool` o subagente.
Verificador	Tool con navegador/búsqueda controlada.	Hosted/web tool o tool propia.	MCP connector, tool propia o Agent SDK.	Tool/MCP/Vertex Search según entorno.
Resultado	JSON con hallazgos, evidencia y acciones.	Output type/schema.	Structured output o contrato de tool.	Schema de salida.
Traza	Eventos por especialista.	Tracing nativo + normalización.	Eventos SDK/logs propios.	ADK evaluation/logging.

La decisión fina:

SI NECESITAS...	DISEÑA ASÍ
Que el especialista tome la conversación completa	Handoff.
Que el coordinador conserve control y solo pida una tarea acotada	Agent as tool.
Que el especialista use navegador o búsqueda	Tool con permisos explícitos y límites.
Que el resultado sea revisable	JSON con `claim`, `evidence_url`, `confidence`, `needs_human_review`.
Que se pueda migrar de SDK	Mantén `AgentSpec`, `ToolSpec` y `TraceEvent` propios.

Una migración contada: lo que de verdad cuesta cambiar de SDK

La portabilidad suena abstracta hasta que llega el día en que hay que cambiar de proveedor, y ese día llega más a menudo de lo que se cree: cambia el precio, sale un modelo mejor en otra casa, una cláusula de cumplimiento obliga a mover datos, o el SDK que elegiste deja de mantenerse. Conviene imaginar dos equipos que parten del mismo punto (un agente que funciona sobre el SDK de un proveedor) y que un día reciben la misma orden: «pásalo al SDK de otro». La diferencia entre lo que les cuesta no es de habilidad; es de cómo dibujaron la frontera meses atrás.

El primer equipo construyó deprisa. Usó las abstracciones del SDK tal cual: el tipo `Agent` del proveedor era su unidad de dominio, sus tools devolvían los objetos que el SDK esperaba, su estado vivía en el session store propietario, y sus trazas eran las que el SDK generaba. Funcionó, y rápido. Pero cuando llega la migración, descubren que casi todo su código «de negocio» está entrelazado con tipos y llamadas del proveedor antiguo. Cambiar de SDK no es sustituir una capa: es reescribir el producto, porque el producto y el SDK eran la misma cosa. Su portabilidad, en los términos del capítulo, era baja, y la factura de la migración es proporcional.

El segundo equipo gastó, al principio, un poco más. Antes de tocar el SDK, definió sus propios contratos: una interfaz `AgentSpec` para describir agentes, un `ToolSpec` para sus herramientas con esquema y permisos, un `TraceEvent` propio para la observabilidad. Su dominio hablaba siempre con esas interfaces; un adaptador fino, y solo el adaptador, traducía a las llamadas concretas del proveedor. Ese sobrecoste inicial pareció innecesario en la demo. En la migración, se revela como la mejor inversión del proyecto: cambiar de proveedor es reescribir un adaptador (unos cientos de líneas bien acotadas) mientras el resto del sistema, sus tests incluidos, no se entera. Su portabilidad era alta, y la migración es un trabajo de días, no de meses.

La moraleja no es «no uses SDKs»: los SDKs ahorran código repetitivo de verdad y conviene aprovecharlos. La moraleja es dónde pones la frontera. Un SDK debería ejecutar una parte del trabajo, no definir tu dominio. La pregunta que distingue a los dos equipos se puede hacer hoy, antes de que exista la migración: si mañana tuviera que cambiar de proveedor, ¿qué porcentaje de mi código tendría que tocar? Si la respuesta es «casi todo», el SDK no es una librería, es el esqueleto de tu producto, y has cedido una decisión estratégica a cambio de ir un poco más rápido al principio. El patrón adaptador, viejo y poco glamuroso, es justo lo que mantiene esa decisión en tus manos.

Cómo lo llevaría a cada SDK

La traducción conceptual sería:

CONTRATO PROPIO	OPENAI AGENTS SDK	CLAUDE AGENT SDK/API	GOOGLE ADK
`AgentSpec`	`Agent(...)`	`query(...)` con configuración, subagente o loop propio.	`Agent(...)` / `LlmAgent(...)`.
`ToolSpec`	`@function_tool`, hosted tool o MCP.	`tools` en Messages API, MCP connector o Agent SDK tools.	Function tools, built-ins, MCP toolbox.
`SessionStore`	`Session` implementation.	Historial propio o Agent SDK runtime.	`SessionService`.
`MemoryStore`	Store propio o sesión/custom.	`CLAUDE.md`, memory tool o store externo.	`MemoryService`.
`TraceEvent`	Tracing nativo + export.	Eventos SDK/logs propios.	Evaluation/logging + observabilidad propia.
`EvalDataset`	Evals/trace grading o harness propio.	Tests/evals propios.	ADK evaluation.

El código de producción debería tener una carpeta parecida a esta:

```
agent_app/
  contracts/
    agent_spec.py
    tool_spec.py
    trace_event.py
  tools/
    revisar_rae.py
    revisar_apa.py
    verificar_fuente.py
  adapters/
    openai_agents.py
    claude_agent.py
    google_adk.py
  evals/
    revision_academica.jsonl
    rubric.yaml
  observability/
    trace_exporter.py
  config/
    agents.yaml
```

La carpeta `contracts` es el centro. Los adaptadores son reemplazables. Si un adaptador crece demasiado, probablemente estás metiendo lógica de dominio dentro del proveedor.

Lo que cada SDK esconde y lo que no deberías dejar que esconda

Un buen SDK te ahorra trabajo. También puede esconder decisiones importantes.

DECISIÓN	PUEDE ESCONDERLA EL SDK	DEBE QUEDAR VISIBLE PARA TI
Cómo se forma el prompt final	Sí.	Context manifest o log de piezas incluidas.
Cuándo se llama una tool	Parcialmente.	Tool call, argumentos, permiso y resultado.
Cómo se guarda sesión	Sí.	Qué entra, qué se compacta y cómo se borra.
Cómo se hace handoff	Sí.	Qué historia recibe el especialista.
Cómo se trazan eventos	Sí.	Export normalizado y retención.
Cómo se reintenta	A veces.	Política de idempotencia y límite.
Cómo se calcula coste	A veces.	Coste por tarea aceptada.
Cómo se evalúa	A veces.	Dataset, métrica y umbral de cambio.

La pregunta que me haría antes de desplegar:

> Si mañana el SDK actual deja de funcionar, ¿qué piezas de mi sistema puedo conservar intactas?

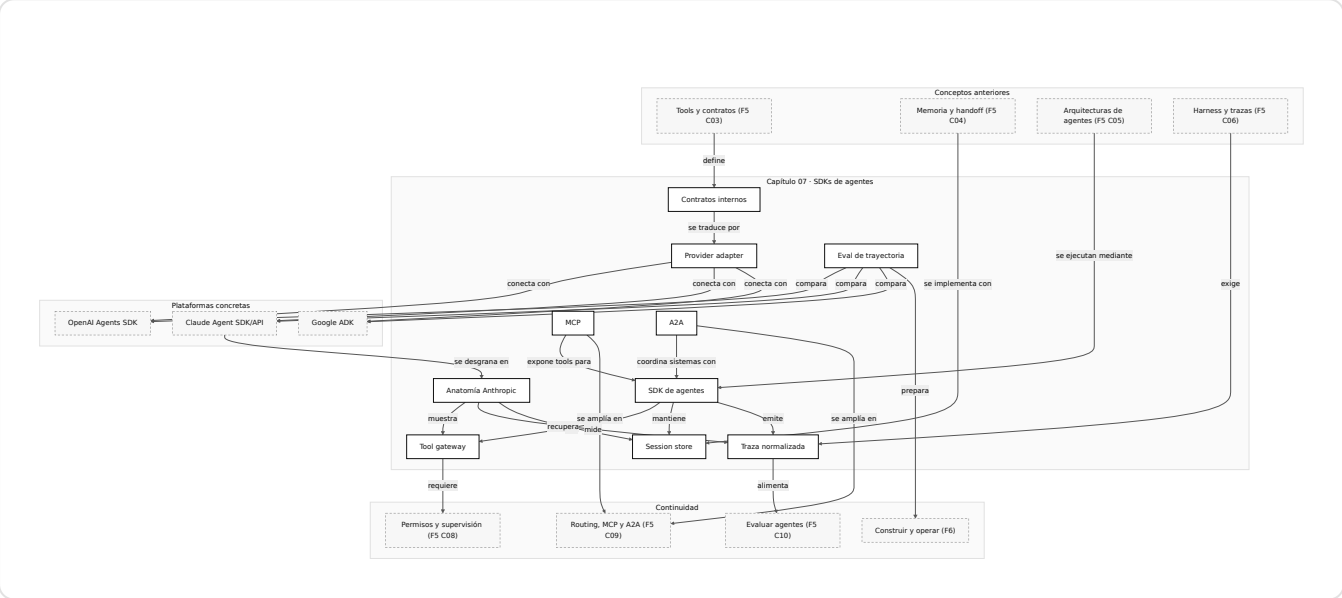
Si la respuesta es “casi ninguna”, el SDK no está integrado: está gobernando el diseño.

Tu turno: calcula tu portabilidad

La integración del facsímil es portable a propósito; ahora mide la tuya. Si ya tienes un agente o un prototipo conectado a un SDK, ábrelo y haz un inventario honesto: ¿qué partes son tuyas (esquemas de tool, lógica de dominio, trazas, tests, estado) y qué partes son tipos y llamadas atados a ese proveedor concreto? Si todavía no tienes nada construido, hazlo sobre el diseño que tienes en mente, que es el mejor momento para decidir dónde va la frontera.

La apuesta es la pregunta que distingue una librería de un esqueleto: si mañana tu empresa te pidiera cambiar de proveedor (porque sube el precio, sale un modelo mejor o lo exige una cláusula de cumplimiento), ¿qué porcentaje de tu código tendrías que tocar? Estima ese número de verdad, aunque sea a ojo. Si la respuesta es «casi todo», no tienes un problema técnico, tienes un riesgo estratégico, y el ejercicio que te llevas es concreto: dibuja la interfaz mínima (`AgentSpec` , `ToolSpec` , `TraceEvent`) que pondría tu dominio en el centro y dejaría el SDK reducido a un adaptador. No hace falta que la implementes hoy; hace falta que sepas cuánto te costaría tu propia migración antes de que alguien te la imponga con prisas.

Cómo encaja todo



Vocabulario aprendido

TÉRMINO	DEFINICIÓN ÚTIL
SDK	Librería y convenciones para usar una plataforma desde código.
Runtime de agente	Capa que ejecuta el bucle entre modelo, tools, estado y resultado.
Adapter	Traducción entre tu contrato interno y el formato de un proveedor.
Capability flag	Marca que indica si un proveedor soporta una capacidad concreta.
Agent as tool	Patrón donde un agente especializado aparece como tool de otro agente.
Handoff	Transferencia de control a otro agente especialista.
Tool gateway	Capa que valida, autoriza, ejecuta y registra tools.
Session store	Almacén de historial o estado conversacional.
Memory store	Almacén de hechos, preferencias o recuerdos reutilizables.
Trace event	Evento observable de una ejecución agentic.
MCP	Protocolo para exponer herramientas y contexto a agentes.
A2A	Protocolo para comunicación entre sistemas agentic.
Vendor lock-in	Dependencia fuerte de una plataforma por acoplar contratos internos a ella.
Context manifest	Recibo de qué piezas entraron al contexto de una llamada.
RunSpec	Contrato de ejecución: entrada, agente, proveedor, presupuesto, estado y salida esperada.
Idempotency key	Identificador que permite repetir una operación sin duplicar su efecto.
Schema drift	Desalineación entre el esquema que esperas y lo que el SDK, modelo o tool devuelve tras cambios.
Eval gate	Prueba que decide si una versión puede avanzar porque cumple trayectoria, calidad y coste.
Eval de trayectoria	Evaluación que revisa pasos, tools, coste y resultado final.

TÉRMINO	DEFINICIÓN ÚTIL
Claude Agent SDK	Runtime de Anthropic construido sobre el arnés de Claude Code para ejecutar agentes desde código.
`query()`	Entrada simple al Agent SDK: manda una tarea y consume mensajes del agente.
Permission callback	Función propia que decide si una tool concreta puede ejecutarse en un contexto concreto.
Hook	Punto de intervención antes o después de ciertos eventos del agente.
Checkpoint	Punto recuperable de una ejecución para continuar o depurar una run larga.

Dónde solía tropezar yo

TROPIEZO	POR QUÉ OCURRE	ANTÍDOTO
Empezar por el tutorial	El ejemplo mínimo parece suficiente.	Escribir primero `AgentSpec`, `ToolSpec` y `TraceEvent`.
Meter lógica de negocio en el adapter	Es rápido al principio.	El adapter solo traduce; el dominio vive fuera.
Confundir sesión con memoria	El SDK guarda historial y parece memoria.	Separar `SessionStore` y `MemoryStore`.
No normalizar trazas	Cada proveedor registra distinto.	Definir eventos comunes antes de comparar.
Usar handoff para todo	Suena elegante delegar.	Usar handoff solo si el especialista debe tomar el control.
Dejar tools demasiado amplias	Es cómodo exponer una función genérica.	Tools pequeñas, efecto declarado y aprobación cuando toque.
Ignorar evaluación de trayectoria	Solo se mira la respuesta final.	Medir steps, tools, coste, estado y salida.

Antes de pasar página

Antes de pasar al capítulo 08, deberías poder responder:

PREGUNTA	SI DUDAS, VUELVE A...
¿Qué diferencia hay entre API de modelo, SDK de cliente y SDK de agentes?	`La definición útil`.
¿Qué piezas forman una integración de SDK completa?	`La anatomía formal de una integración`.
¿Qué ofrece OpenAI Agents SDK que no es solo una llamada de modelo?	`OpenAI Agents SDK: runtime opinado para agentes`.
¿Por qué Anthropic tiene dos niveles distintos: Messages API y Claude Agent SDK?	`Anthropic: Messages API, Claude Agent SDK y Claude Code`.
¿Cómo se descompone una ejecución del SDK de Anthropic?	`Anatomía del SDK de Anthropic`.
¿Qué diferencia hay entre permisos, hooks y tools permitidas?	`Permisos: la parte que no se debe improvisar`.
¿Qué aporta Google ADK en sesiones, memoria y evaluación?	`Google ADK: framework de agentes, sesiones, memoria y evaluación`.
¿Qué diferencia hay entre MCP y A2A?	`MCP y A2A: no son lo mismo que un SDK`.
¿Cuándo usar API directa, SDK de agente, framework, MCP o A2A?	`Qué le faltaba al capítulo`.
¿Qué debe llevar una run para ser depurable?	`Ingeniería de producción`.
¿Por qué conviene diseñar contratos propios antes de elegir proveedor?	Practícalo en el cuaderno del facsímil.
¿Qué debe quedar visible aunque el SDK lo automatice?	`Lo que cada SDK esconde y lo que no deberías dejar que esconda`.

Para saber más

- Agent2Agent Protocol. (2026). *Specification*. https://google-a2a.github.io/A2A/specification/
- Anthropic. (2026). *Claude Agent SDK overview*. https://code.claude.com/docs/en/agent-sdk/overview
- Anthropic. (2026). *Claude Agent SDK quickstart*. https://code.claude.com/docs/en/agent-sdk/quickstart

- Anthropic. (2026). *Claude Agent SDK: Agent loop*. https://code.claude.com/docs/en/agent-sdk/agent-loop
- Anthropic. (2026). *Claude Agent SDK: Checkpointing*. https://code.claude.com/docs/en/agent-sdk/checkpointing
- Anthropic. (2026). *Claude Agent SDK: Cost tracking*. https://code.claude.com/docs/en/agent-sdk/cost-tracking
- Anthropic. (2026). *Claude Agent SDK: Hooks*. https://code.claude.com/docs/en/agent-sdk/hooks
- Anthropic. (2026). *Claude Agent SDK: Observability with OpenTelemetry*. https://code.claude.com/docs/en/agent-sdk/observability
- Anthropic. (2026). *Claude Agent SDK: Permissions*. https://code.claude.com/docs/en/agent-sdk/permissions
- Anthropic. (2026). *How to implement tool use*. https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/implement-tool-use
- Anthropic. (2026). *MCP connector*. https://platform.claude.com/docs/en/agents-and-tools/mcp-connector
- Anthropic. (2026). *Streaming Messages*. https://platform.claude.com/docs/en/build-with-claude/streaming
- Anthropic. (2026). *Using the Messages API*. https://platform.claude.com/docs/en/build-with-claude/working-with-messages
- Ellram, L. M. (1995). Total cost of ownership: an analysis approach for purchasing. *International Journal of Physical Distribution & Logistics Management*, 25(8), 4-23. https://doi.org/10.1108/09600039510099928
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Google. (2026). *ADK with Agent2Agent Protocol*. https://adk.dev/a2a/
- Google. (2026). *Agent Development Kit: Agents*. https://adk.dev/agents/
- Google. (2026). *Agent Development Kit: Memory*. https://adk.dev/sessions/memory/
- Google. (2026). *Agent Development Kit: Technical overview*. https://adk.dev/get-started/about/
- Google. (2026). *Agent Development Kit: Tools*. https://adk.dev/tools/
- Google. (2026). *Agent Development Kit: Why Evaluate Agents*. https://adk.dev/evaluate/

- Martin, R. C. (2002). *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall.
- Model Context Protocol. (2026). *Specification*. https://modelcontextprotocol.io/specification
- OpenAI. (2026). *Agents SDK*. https://developers.openai.com/api/docs/guides/agents
- OpenAI. (2026). *Agents SDK JS: Model Context Protocol*. https://openai.github.io/openai-agents-js/guides/mcp/
- OpenAI. (2026). *Agents SDK JS: Sessions*. https://openai.github.io/openai-agents-js/guides/sessions/
- OpenAI. (2026). *Agents SDK JS: Tools*. https://openai.github.io/openai-agents-js/guides/tools
- OpenAI. (2026). *Agents SDK: Agents*. https://openai.github.io/openai-agents-python/agents/
- OpenAI. (2026). *Agents SDK: Handoffs*. https://openai.github.io/openai-agents-python/handoffs/
- OpenAI. (2026). *Agents SDK: Tracing*. https://openai.github.io/openai-agents-python/tracing/
- Patil, S. G., Zhang, T., Wang, X., & Gonzalez, J. E. (2023). Gorilla: Large Language Model Connected with Massive APIs. arXiv. https://doi.org/10.48550/arXiv.2305.15334
- Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., & Sun, M. (2023). ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs. arXiv. https://doi.org/10.48550/arXiv.2307.16789
- Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv. https://doi.org/10.48550/arXiv.2302.04761

En resumen

IDEA	QUÉ TE LLEVAS
El SDK no es la arquitectura completa.	El proveedor ejecuta una parte; tu producto debe controlar contratos, estado, tools, trazas y evaluación.
OpenAI, Anthropic y Google ADK resuelven capas distintas.	OpenAI ofrece un runtime opinado, Anthropic combina API transparente y Agent SDK, Google ADK integra agentes, memoria y evaluación.
La portabilidad se diseña antes de instalar paquetes.	`AgentSpec`, `ToolSpec`, `TraceEvent` y capability flags evitan que el SDK gobierne el dominio.
MCP y A2A son fronteras, no atajos conceptuales.	MCP expone tools y contexto; A2A coordina sistemas agentic.
La integración buena se mide.	Evalúa trayectoria, coste, tools, estado y salida, no solo que la demo responda.

Notas

1. Martin, R. C. (2002). *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall. Define métricas de acoplamiento e inestabilidad de un componente según sus dependencias. ↩

2. OpenAI. (2026). *Agents SDK: Agents*. https://openai.github.io/openai-agents-python/agents/. Consultado el 10 de junio de 2026. ↩

3. OpenAI. (2026). *Agents SDK*. https://developers.openai.com/api/docs/guides/agents. Consultado el 10 de junio de 2026. ↩

4. OpenAI. (2026). *Agents SDK: Tracing*. https://openai.github.io/openai-agents-python/tracing/. Consultado el 10 de junio de 2026. ↩

5. OpenAI. (2026). *Agents SDK JS: Tools*. https://openai.github.io/openai-agents-js/guides/tools. Consultado el 10 de junio de 2026. ↩

6. OpenAI. (2026). *Agents SDK: Handoffs*. https://openai.github.io/openai-agents-python/handoffs/. Consultado el 10 de junio de 2026. ↩

7. OpenAI. (2026). *Agents SDK JS: Sessions*. https://openai.github.io/openai-agents-js/guides/sessions/. Consultado el 10 de junio de 2026. ↩

8. OpenAI. (2026). *Agents SDK JS: Model Context Protocol*. https://openai.github.io/openai-agents-js/guides/mcp/. Consultado el 10 de junio de 2026. ↩

9. Anthropic. (2026). *Using the Messages API*. https://platform.claude.com/docs/en/build-with-claude/working-with-messages. Consultado el 10 de junio de 2026. ↩

10. Anthropic. (2026). *Claude Agent SDK overview*. https://code.claude.com/docs/en/agent-sdk/overview. Consultado el 10 de junio de 2026. ↩

11. Anthropic. (2026). *Claude Agent SDK quickstart*. https://code.claude.com/docs/en/agent-sdk/quickstart. Consultado el 10 de junio de 2026. ↩

12. Anthropic. (2026). *How to implement tool use*. https://docs.anthropic.com/en/docs/agents-and-tools/tool-use/implement-tool-use. Consultado el 10 de junio de 2026. ↩

13. Anthropic. (2026). *MCP connector*. https://platform.claude.com/docs/en/agents-and-tools/mcp-connector. Consultado el 10 de junio de 2026. ↩

14. Anthropic. (2026). *Using the Messages API*. https://platform.claude.com/docs/en/build-with-claude/working-with-messages. Consultado el 10 de junio de 2026. ↩

15. Anthropic. (2026). *Claude Agent SDK quickstart*. https://code.claude.com/docs/en/agent-sdk/quickstart. Consultado el 10 de junio de 2026. ↩

16. Anthropic. (2026). *Claude Agent SDK: Agent loop*. https://code.claude.com/docs/en/agent-sdk/agent-loop. Consultado el 10 de junio de 2026. ↩

17. Saltzer, J. H. y Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278-1308. https://doi.org/10.1109/PROC.1975.9939 Formula la mediación completa: cada acceso se comprueba antes de concederse. ↩

18. Anthropic. (2026). *Claude Agent SDK: Permissions*. https://code.claude.com/docs/en/agent-sdk/permissions. Consultado el 10 de junio de 2026. ↩

19. Anthropic. (2026). *Claude Agent SDK: Hooks*. https://code.claude.com/docs/en/agent-sdk/hooks. Consultado el 10 de junio de 2026. ↩

20. Anthropic. (2026). *Streaming Messages*. https://platform.claude.com/docs/en/build-with-claude/streaming. Consultado el 10 de junio de 2026. ↩

21. Anthropic. (2026). *Claude Agent SDK: Checkpointing*. https://code.claude.com/docs/en/agent-sdk/checkpointing. Consultado el 10 de junio de 2026. ↩

22. Anthropic. (2026). *Claude Agent SDK: Cost tracking*. https://code.claude.com/docs/en/agent-sdk/cost-tracking. Consultado el 10 de junio de 2026. ↩

23. Anthropic. (2026). *Claude Agent SDK: Observability with OpenTelemetry*. https://code.claude.com/docs/en/agent-sdk/observability. Consultado el 10 de junio de 2026. ↩

24. Google. (2026). *Agent Development Kit: Agents*. https://adk.dev/agents/. Consultado el 10 de junio de 2026. ↩

25. Google. (2026). *Agent Development Kit: Technical overview*. https://adk.dev/get-started/about/. Consultado el 10 de junio de 2026. ↩

26. Google. (2026). *Agent Development Kit: Memory*. https://adk.dev/sessions/memory/. Consultado el 10 de junio de 2026. ↩

27. Google. (2026). *Agent Development Kit: Tools*. https://adk.dev/tools/. Consultado el 10 de junio de 2026. ↩

28. Google. (2026). *Agent Development Kit: Why Evaluate Agents*. https://adk.dev/evaluate/. Consultado el 10 de junio de 2026. ↩

29. Google. (2026). *ADK with Agent2Agent Protocol*. https://adk.dev/a2a/. Consultado el 10 de junio de 2026. ↩

30. Model Context Protocol. (2026). *Specification*. https://modelcontextprotocol.io/specification. Consultado el 10 de junio de 2026. ↩

31. Agent2Agent Protocol. (2026). *Specification*. https://google-a2a.github.io/A2A/specification/. Consultado el 10 de junio de 2026. ↩

32. Gamma, E., Helm, R., Johnson, R. y Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. Define el patrón adaptador, que envuelve una interfaz externa tras una propia para desacoplar el sistema del proveedor. ↩

33. Ellram, L. M. (1995). Total cost of ownership: an analysis approach for purchasing. *International Journal of Physical Distribution & Logistics Management*, 25(8), 4-23. https://doi.org/10.1108/09600039510099928 El coste real es la suma de todos los componentes que hay que sostener. ↩

34. OpenAI. (2026). *Agents SDK: Agents*. https://openai.github.io/openai-agents-python/agents/. Consultado el 10 de junio de 2026. ↩

35. OpenAI. (2026). *Agents SDK: Tracing*. https://openai.github.io/openai-agents-python/tracing/. Consultado el 10 de junio de 2026. ↩

36. Anthropic. (2026). *Using the Messages API*. https://platform.claude.com/docs/en/build-with-claude/working-with-messages. Consultado el 10 de junio de 2026. ↩

37. Anthropic. (2026). *Claude Agent SDK overview*. https://code.claude.com/docs/en/agent-sdk/overview. Consultado el 10 de junio de 2026. ↩

38. Google. (2026). *Agent Development Kit: Technical overview*. https://adk.dev/get-started/about/. Consultado el 10 de junio de 2026. ↩

39. Google. (2026). *Agent Development Kit: Why Evaluate Agents*. https://adk.dev/evaluate/. Consultado el 10 de junio de 2026. ↩