EDICIÓN PÚBLICA MATERIAL INDEPENDIENTE · NO ES DOCUMENTACIÓN OFICIAL

Claude *Agent SDK* en Python: arquitectura, patrones y migración.

Una lectura guiada para entender por qué Anthropic externalizó el motor de Claude Code como librería, qué cambia frente al SDK estándar, cuándo conviene migrar y cuándo es over-engineering. Verificada contra la documentación oficial de Anthropic con correcciones técnicas al PDF de respaldo original.

TIEMPO DE LECTURA	NIVEL	PARTES	PATRONES
≈ 60 min	Intermedio · Python	8 + Anexos	5 + variantes

CONVENCIÓN DEL DOCUMENTO

Esta guía distingue dos arquitecturas a lo largo de toda la lectura:

- **SDK Estándar** — `anthropic.Anthropic()` con API keys tradicionales. El desarrollador controla el bucle del agente manualmente.
- **Agent SDK** — `claude_agent_sdk` con `query()` o `ClaudeSDKClient`. El bucle vive dentro del SDK; el desarrollador configura el entorno.

Cuando algo aplique a solo una, se indica explícitamente. Toda la información está verificada contra documentación oficial de Anthropic vigente al 15 de mayo de 2026 (URLs al final del documento).

AVISO · MATERIAL INDEPENDIENTE

Este es material de estudio **independiente; no es documentación oficial de Anthropic**. Ante cualquier discrepancia, prevalece la documentación oficial (`platform.claude.com/docs`, `code.claude.com/docs`).

Los nombres de proyectos, empresas y datos de ejemplo son **ficticios e ilustrativos**. Las marcas mencionadas pertenecen a sus respectivos dueños. Elaborado por **Francisco José Barros Cruz** con asistencia de IA.

§ TABLA DE CONTENIDOS

Índice.

01 PARTE 01

Fundamentos conceptuales.

El ecosistema de desarrollo con modelos de lenguaje ha transitado rápidamente desde la generación de texto estático hacia la orquestación de flujos de trabajo complejos y autónomos. Para comprender el valor estratégico de la herramienta, es imperativo establecer la diferencia entre una *API conversacional* y un *entorno de ejecución agentizado*.

Historia y motivación de la arquitectura

El Claude Agent SDK nació originalmente como el motor interno de **Claude Code**, la herramienta de interfaz de línea de comandos (CLI) de Anthropic diseñada para el desarrollo de software interactivo. En **septiembre de 2025**, Anthropic renombró el paquete de *"Claude Code SDK"* a *"Claude Agent SDK"*. Este cambio de nomenclatura reflejó una realidad técnica ineludible: el entorno de ejecución (*runtime*) construido para editar código era lo suficientemente robusto y generalizable para operar flujos de trabajo agenticos en dominios legales, de investigación, análisis financiero y soporte automatizado.

La motivación principal de Anthropic para externalizar este motor como una librería pública fue **eliminar el trabajo redundante** (*boilerplate*). Con el SDK estándar, los desarrolladores se veían obligados a programar manualmente el bucle del agente (*agent loop*). Esto implicaba gestionar el crecimiento del historial de mensajes, analizar la razón de detención (`stop_reason`), ejecutar herramientas locales, formatear los resultados y devolverlos al modelo, todo mientras se gestionaba la saturación de la ventana de contexto. El Agent SDK absorbe toda esta orquestación de bajo nivel, permitiendo al ingeniero **enfocarse exclusivamente en la lógica de negocio y en la definición de límites de seguridad**.

"LLM como Respondedor" vs. "LLM como Agente que Opera"

La distinción arquitectónica fundamental radica en **la inversión del flujo de control**.

Bajo el paradigma del *"LLM como Respondedor"* (SDK Estándar), el código Python del desarrollador es el controlador maestro. Se realiza una solicitud al modelo, este sugiere una acción a través de un esquema JSON, la ejecución retorna a Python, la función se ejecuta y Python vuelve a invocar al modelo. El LLM actúa como un componente sin estado (*stateless*) consultado de forma puramente reactiva.

Por el contrario, en el paradigma del *"LLM como Agente que Opera"* (Agent SDK), el bucle de ejecución reside dentro del SDK. Se le entrega al modelo un objetivo, un conjunto de herramientas y límites de costo. El LLM planifica, llama a múltiples herramientas (incluso en paralelo), recibe los resultados de manera interna, evalúa si la tarea está completa y solo devuelve el control al hilo de Python cuando la tarea ha finalizado, ha agotado su presupuesto o requiere intervención humana. **Esta distinción importa** porque permite delegar la micro-toma de decisiones al modelo, reduciendo la latencia de red y la complejidad del código del lado del cliente.

SDK ESTÁNDAR

LLM como Respondedor

El código Python controla el flujo. El modelo es consultado reactivamente sin estado.

1. Python envía solicitud al modelo
2. El modelo sugiere acción vía esquema JSON
3. Python ejecuta la función localmente
4. Python re-invoca al modelo con el resultado
5. El LLM es *stateless*, reactivo

AGENT SDK

LLM como Agente que Opera

El bucle vive dentro del SDK. Se entrega objetivo + herramientas + límites; el modelo planifica y resuelve.

1. Se entrega objetivo + herramientas + límites
2. El LLM planifica internamente
3. Llama múltiples herramientas (incluso en paralelo)
4. Recibe resultados y evalúa internamente
5. Solo retorna a Python al terminar / agotar presupuesto

Relación entre Claude Code y Agent SDK

Es un error común asumir que el SDK es simplemente una envoltura sobre la herramienta de terminal. De hecho, **la relación es inversa**: Claude Code y el Claude Agent SDK comparten exactamente el mismo motor de ejecución y el mismo bucle de agente.

El análisis de la arquitectura base revela que ambas herramientas convergen en una única función asíncrona denominada `queryLoop()`. La diferencia radica en la **capa de interfaz**: Claude Code envuelve este motor en una experiencia interactiva para terminales, mientras que el Agent SDK lo expone como una librería programable en **Python y TypeScript** para ser incrustada en servidores, tuberías de datos (*pipelines*) y tareas asíncronas.

Modelo mental del procesamiento de solicitudes

El procesamiento de una solicitud en el Agent SDK sigue un ciclo continuo y determinista, organizado en tres fases narrativas:

La fase inicial comienza con la recepción del prompt. El SDK ensambla el contexto cargando el prompt del sistema, las definiciones de herramientas registradas, el historial de la sesión y cualquier instrucción a nivel de proyecto (como los archivos `CLAUDE.md`). En este punto, el SDK emite un mensaje de inicialización que contiene los metadatos de la sesión.

La fase de evaluación sigue inmediatamente. El modelo analiza el estado actual del entorno y determina la mejor forma de proceder. Decide si la respuesta requiere texto directo, la invocación de herramientas, o una combinación de ambas. Si se solicitan herramientas, el flujo se detiene momentáneamente para ejecutar los

eventos de interceptación (Hooks) del tipo `PreToolUse`. Si los validadores aprueban la acción, el SDK ejecuta la herramienta en el sistema local y recolecta los resultados. Inmediatamente después, se ejecutan los eventos `PostToolUse` para auditoría o inyección de estado.

El bucle cerrado repite el proceso de evaluación y ejecución hasta que el modelo determina que la tarea está completa y emite una respuesta de texto libre de invocaciones a herramientas. Finalmente, el SDK consolida la ejecución devolviendo un *mensaje de resultado* (`ResultMessage`) que encapsula el texto final, la telemetría de tokens consumidos, el costo financiero de la operación y el identificador de la sesión para futuras retomas.

> **PRESCRIPCIÓN ARQUITECTÓNICA**
>
> Comprender este modelo mental es vital. Cuando utilices el Agent SDK, **no intentes micro-gestionar el flujo de ejecución**. Diseña herramientas robustas con manejo de errores interno y define límites claros, pero permite que el motor `queryLoop()` resuelva la heurística de navegación de la tarea.

02 PARTE 02 Diferencias técnicas concretas.

La adopción del Agent SDK transforma radicalmente la estructura del código base. A continuación se presenta una comparación exhaustiva de las diferencias en implementación para las mecánicas más comunes.

Comparación del bucle de herramientas (Tool Loop)

La diferencia más drástica entre ambos ecosistemas se manifiesta en la invocación de funciones (*function calling*). Con el SDK estándar, la gestión del bucle recae completamente en el desarrollador. El siguiente código ilustra un bucle manual robusto utilizando la API de mensajes directa:

PYTHON · SDK ESTÁNDAR

```python
import os
from anthropic import Anthropic

class="tok-kw">def class="tok-fn">execute_weather_tool(location: str) → str:
    if "santiago" in location.lower():
        return "22 degrees, sunny"
    return "Unknown weather"

class="tok-kw">def class="tok-fn">manual_agent_loop(prompt: str) → str:
    client = Anthropic(api_key=os.environ.get("ANTHROPIC_API_KEY"))
    messages = [{"role": "user", "content": prompt}]
    tools = [{
        "name": "get_weather",
        "description": "Get weather for a given location",
        "input_schema": {
            "type": "object",
            "properties": {"location": {"type": "string"}},
            "required": ["location"]
        }
    }]

    # Bucle manual de control de flujo
    while True:
        response = client.messages.create(
            model="claude-sonnet-4-6",
            max_tokens=1024,
            messages=messages,
            tools=tools
        )
        messages.append({"role": "assistant", "content": response.content})

        if response.stop_reason ≠ "tool_use":
            return response.content[0].text

        tool_results = []
        for block in response.content:
            if block.type == "tool_use" and block.name == "get_weather":
                location = block.input.get("location", "")
                result = execute_weather_tool(location)
                tool_results.append({
                    "type": "tool_result",
                    "tool_use_id": block.id,
                    "content": result
                })

        messages.append({"role": "user", "content": tool_results})

if __name__ == "__main__":
    print(manual_agent_loop("What is the weather in Santiago?"))
```

El mismo comportamiento con Agent SDK

El mismo comportamiento se implementa externalizando el bucle y la definición de esquemas a través de decoradores y servidores MCP (Model Context Protocol) locales:

PYTHON · AGENT SDK

```python
import asyncio
from typing import Dict, Any
from claude_agent_sdk import (
    query, ClaudeAgentOptions, tool, create_sdk_mcp_server,
    AssistantMessage, TextBlock
)

# El decorador auto-genera el esquema JSON a partir de los type hints
@tool("get_weather", "Get weather for a given location", {"location": str})
async def get_weather(args: Dict[str, Any]) -> Dict[str, Any]:
    location = args.get("location", "")
    result = "22 degrees, sunny" if "santiago" in location.lower() else "Unknown"
    return {"content": [{"type": "text", "text": result}]}

# Empaquetado como servidor MCP local in-process
weather_server = create_sdk_mcp_server(
    name="weather_srv",
    version="1.0.0",
    tools=[get_weather]
)

async def automatic_agent_loop(prompt: str) -> None:
    options = ClaudeAgentOptions(
        model="claude-sonnet-4-6",
        mcp_servers={"weather": weather_server},
        allowed_tools=["mcp__weather_srv__get_weather"]
    )
    # El iterador asíncrono gestiona las iteraciones hasta completar la tarea
    async for message in query(prompt=prompt, options=options):
        if isinstance(message, AssistantMessage):
            for block in message.content:
                if isinstance(block, TextBlock):
                    pass  # registrar progreso intermedio
        elif hasattr(message, "result"):
            print(message.result)

if __name__ == "__main__":
    asyncio.run(automatic_agent_loop("What is the weather in Santiago?"))
```

Análisis comparativo de capacidades

El comportamiento del sistema difiere profundamente según el SDK utilizado. La siguiente tabla sintetiza cómo se manejan las operaciones críticas en ambos paradigmas:

CAPACIDAD	SDK ESTÁNDAR	AGENT SDK
Manejo de permisos	Inexistente nativamente. La lógica del desarrollador evalúa el JSON devuelto y decide ejecutar o abortar.	Cinco modos de `permission_mode`. Listas blancas vía `allowed_tools` (auto-aprueba) y listas negras vía `disallowed_tools` (siempre bloquea).
Acceso a File System	Requiere herramientas en Python usando `os` / `pathlib`, bloqueos y codificación manual.	Herramientas pre-entrenadas nativas: **Read, Write, Edit, Glob, Grep**. Acceso delimitado por `cwd`.
Ejecución de Bash	Usar `subprocess.run`, manejar timeouts, stderr y sanitización de comandos.	Herramienta **Bash** nativa empaquetada, ejecutada en sandbox configurable.
Búsqueda Web	Integrar BeautifulSoup, Playwright o APIs pagadas de terceros.	**WebFetch** y **WebSearch** nativas, con parseo HTML→texto transparente.
Manejo de errores	Capturar excepciones, formatear traceback como texto y reinyectarlo al LLM.	Captura fallos automáticamente, los convierte en `ResultMessage`, el modelo itera y se auto-corrige.
Persistencia	La aplicación anfitriona almacena el array de mensajes en memoria o BD externa.	Estado en disco (`~/.claude/projects/<encoded-cwd>/<session-id>.jsonl`). Tres modos de retoma: `continue_conversation`, `resume=session_id`, `fork_session=True`.
Sub-agentes	Instanciar clientes independientes, particionar contexto manualmente, orquestar secuencialmente.	Herramienta **Agent** + clase `AgentDefinition`. Contexto aislado del orquestador. Trazabilidad vía `parent_tool_use_id`.

PRESCRIPCIÓN TÉCNICA

El **SDK Estándar** es la arquitectura correcta cuando el modelo actúa como una tubería de transformación de texto predecible sin efectos secundarios (*side effects*) sobre el sistema. El **Agent SDK** resulta indispensable cuando la solución demanda interacción autónoma con recursos del sistema operativo, acceso a internet no estructurado o flujos de razonamiento de múltiples turnos.

Los cinco modos de `permission_mode`

El SDK Python expone **cinco modos exactos** de permisos, no tres como mencionan algunas fuentes. El PDF de respaldo lista `"auto"` que **no existe en Python** (solo en TypeScript SDK).

MODO	COMPORTAMIENTO
`default`	Comportamiento estándar. No hay auto-aprobaciones; herramientas no resueltas activan el callback `can_use_tool`.
`acceptEdits`	Auto-aprueba operaciones de archivo (Edit, Write) y comandos filesystem (`mkdir`, `rm`, `mv`, `cp`, `sed`). Solo dentro de `cwd` o `add_dirs`.
`plan`	Modo planificación: solo herramientas de lectura. Claude analiza y propone sin editar archivos. Usa `AskUserQuestion` si necesita aclarar.
`dontAsk`	Cualquier cosa no pre-aprobada por `allowed_tools` o reglas se deniega sin preguntar. `can_use_tool` nunca se llama.
`bypassPermissions`	Bypass total. ⚠ **Hooks y deny rules aún ejecutan** y pueden bloquear. Acceso autónomo completo — usar con extrema precaución.

HERENCIA EN SUB-AGENTES

Cuando el orquestador padre usa `bypassPermissions`, `acceptEdits` o `auto`, **todos los sub-agentes heredan ese modo automáticamente** y no puede overridearse per-subagent. Como los sub-agentes pueden tener prompts diferentes y comportamiento menos restringido, heredar `bypassPermissions` les otorga acceso autónomo completo al sistema sin prompts de aprobación.

Orden de evaluación de permisos (5 pasos)

Cuando Claude solicita una herramienta, el SDK evalua los permisos en este orden estricto:

1. **Hooks.** Ejecutan primero. Un hook puede denegar directamente o pasar. Un hook que retorna `allow` *no* salta los pasos de deny ni ask siguientes.
2. **Deny rules.** Se chequea `disallowed_tools` y reglas en `settings.json`. Si una regla match, la herramienta es bloqueada — **incluso en modo `bypassPermissions`**.
3. **Permission mode.** Aplica el modo activo. `bypassPermissions` aprueba todo lo que llega; `acceptEdits` aprueba operaciones de archivo; otros modos siguen al paso 4.
4. **Allow rules.** Se chequea `allowed_tools` y reglas en settings. Si match, herramienta aprobada.
5. **can_use_tool callback.** Si nada lo resolvió, se llama tu callback custom. **En modo `dontAsk`, este paso se salta y la herramienta se deniega.**

ALLOWED_TOOLS VS DISALLOWED_TOOLS: DISTINCIÓN CRÍTICA

`allowed_tools` **NO restringe a Claude a usar solo esas herramientas.** Solo las pre-aprueba para que no se pregunte. Las herramientas no listadas siguen existiendo y caen a `permission_mode` y `can_use_tool`. Para bloquear herramientas de verdad: usa `disallowed_tools`.

Trampa común: `allowed_tools=["Read"]` + `permission_mode="bypassPermissions"` **no bloquea Bash o Write.** Como `bypassPermissions` aprueba todo lo que llega al paso 3, las herramientas no listadas igual se aprueban. Para un agente verdaderamente acotado: combina `allowed_tools` con `permission_mode="dontAsk"`.

03 PARTE 03 Anatomía de un agente.

El desarrollo con el Agent SDK requiere abandonar el paradigma de envío de mensajes en favor de la *configuración de entornos*. Un agente se materializa a través de configuraciones de seguridad, definición de herramientas MCP y la interceptación del ciclo de vida mediante Hooks.

Dos interfaces: `query()` vs `ClaudeSDKClient`

La SDK Python expone dos formas de interactuar con Claude. **Ambas soportan custom tools y hooks**; la diferencia está en el manejo de sesión y ciclo de vida.

CORRECCIÓN RESPECTO A V1.1

La v1.1 (y el PDF de respaldo) implicaban que `query()` no soporta hooks ni custom tools. Es **incorrecto según la documentación oficial actual** (`platform.claude.com/docs/en/agent-sdk/python`). La tabla oficial confirma que ambas interfaces soportan **Hooks ✓** y **Custom Tools ✓**. La diferencia real es continuidad de sesión, soporte de interrupciones, y control de lifecycle.

CAPACIDAD	`QUERY()`	`CLAUDESDKCLIENT`
Sesión	Nueva cada invocación	Reutiliza la misma sesión entre `client.query()` sucesivas
Conversación	Single exchange	Múltiples exchanges en el mismo contexto
Conexión	Automática	Control manual (`connect()`, `disconnect()`)
Streaming Input	✓	✓
Interrupciones mid-task	✗	✓ vía `client.interrupt()`
Hooks	✓	✓
Custom Tools (MCP)	✓	✓
Cambiar `permission_mode` dinámicamente	✗	✓ vía `set_permission_mode()`
Cambiar modelo mid-sesión	✗	✓ vía `set_model()`
Continuar conversación	✗ (nueva sesión c/llamada)	✓ mantiene contexto
Caso de uso	Tareas one-off, scripts simples, cron jobs	Conversaciones continuas, chat UIs, REPLs

Estructura mínima funcional

El punto de entrada más ágil es la función `query()`, la cual inicializa un entorno temporal y retorna un generador asíncrono. La clase `ClaudeAgentOptions` es la estructura de datos que rige las *leyes físicas* de dicho entorno.

PYTHON · AGENTE MÍNIMO

```python
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions

async class="tok-kw">def class="tok-fn">minimal_agent() → None:
    options = ClaudeAgentOptions(
        system_prompt="Eres un agente de sistema. Diagnostica el estado del servidor.",
        allowed_tools=["Bash"],         # Acceso directo a terminal
        permission_mode="acceptEdits",  # Ejecución sin confirmación manual
        cwd="/tmp/agent_workspace"      # Aísla la ejecución a un directorio seguro
    )
    async for message in query(
        prompt="Ejecuta 'uname -a' y 'df -h' y resume el estado.",
        options=options
    ):
        if hasattr(message, "result"):
            print(f"Reporte del Sistema:\n{message.result}")

if __name__ == "__main__":
    asyncio.run(minimal_agent())
```

Configuración de `system_prompt`: defaults importan

Si no especificas `system_prompt`, el SDK usa un **prompt mínimo** que cubre solamente tool calling — omite las guías de Claude Code, su estilo de respuesta, y contexto de proyecto. Esto difiere de `claude -p`, que sí usa el preset completo por defecto. Hay tres formas de configurar el system prompt:

CONFIGURACIÓN	RESULTADO
Omitir / None	Prompt mínimo: solo tool calling, sin guías Claude Code, sin contexto del directorio. **⚠ No es lo que esperas si vienes de la CLI.**
String custom	El SDK envía exactamente lo que escribiste. Pierdes todas las guías y herramientas Claude Code.
Preset dict	`{"type": "preset", "preset": "claude_code"}` activa el prompt completo (guías de código, herramientas, contexto). Añade `"append": "..."` para extender.

PYTHON · TRES FORMAS

```python
# Forma 1: prompt mínimo (default cuando omites)
options = ClaudeAgentOptions()  # SDK envía solo tool calling support

# Forma 2: string custom (reemplaza todo)
options = ClaudeAgentOptions(
    system_prompt="Eres un investigador financiero senior..."
)

# Forma 3: preset claude_code completo + append
options = ClaudeAgentOptions(
    system_prompt={
        "type": "preset",
        "preset": "claude_code",
        "append": "Siempre incluye type hints en código Python."
    }
)
```

Settings filesystem: `setting_sources`

Por defecto, **el SDK no carga ningún settings del filesystem:** ni `~/.claude/settings.json`, ni `.claude/settings.json` del proyecto, ni `CLAUDE.md`. Esto provee aislamiento para apps SDK-only. Si necesitas que carguen, hay que especificar `setting_sources` explícitamente.

VALOR	UBICACIÓN	PARA QUÉ SIRVE
`"user"`	`~/.claude/settings.json`	Settings globales del usuario
`"project"`	`.claude/settings.json` + `CLAUDE.md`	Settings compartidos del proyecto (en git). **Requerido para cargar CLAUDE.md.**
`"local"`	`.claude/settings.local.json`	Settings locales del proyecto (gitignored)

Precedencia (mayor a menor): `local` > `project` > `user`. Las opciones programáticas (como `agents=`, `allowed_tools=`) siempre overridean filesystem.

PYTHON · CARGAR CLAUDE.MD

```python
# Sin setting_sources: NO se carga CLAUDE.md
options = ClaudeAgentOptions(
    system_prompt={"type": "preset", "preset": "claude_code"}
)  # ¡CLAUDE.md ignorado!

# Con setting_sources=[P4P]: sí se carga
options = ClaudeAgentOptions(
    system_prompt={"type": "preset", "preset": "claude_code"},
    setting_sources=["project"]   # Carga CLAUDE.md del directorio
)

# Comportamiento legacy (SDK v0.0.x): cargar todo
options = ClaudeAgentOptions(
    setting_sources=["user", "project", "local"]
)
```

Herramientas personalizadas (In-Process MCP)

Las herramientas personalizadas se implementan como Servidores del Protocolo de Contexto de Modelos (MCP) que corren en el mismo proceso de memoria (*in-process*). Esto elimina la latencia de red y simplifica drásticamente el despliegue. El decorador `@tool` toma el nombre, la descripción y un diccionario tipado de argumentos, generando el esquema JSON de forma automática.

PYTHON · MCP LOCAL

```python
from typing import Dict, Any
from claude_agent_sdk import tool, create_sdk_mcp_server, ClaudeAgentOptions

@tool("query_bank_db", "Busca transacciones bancarias por ID tributario", {"tax_id": str})
async class="tok-kw">def class="tok-fn">query_bank_db(args: Dict[str, Any]) -> Dict[str, Any]:
    tax_id_query = args.get("tax_id", "")
    # Conexión a BD interna (SQLAlchemy, asyncpg, etc.)
    data = f"Transacción reciente para {tax_id_query}: $45.000 en Supermercado"
    return {"content": [{"type": "text", "text": data}]}

# Se consolida la herramienta en un servidor MCP local
banking_mcp = create_sdk_mcp_server(
    name="banking_core",
    version="1.0.0",
    tools=[query_bank_db]
)

options = ClaudeAgentOptions(
    mcp_servers={"banking": banking_mcp},
    allowed_tools=["mcp__banking_core__query_bank_db"]
)
```

Sub-agentes: aislamiento del contexto

La orquestación de sub-agentes resuelve el problema de **polución del contexto**. Un orquestador principal puede invocar a un especialista sin absorber las transcripciones masivas que este genera durante su trabajo. Los sub-agentes se definen mediante la clase `AgentDefinition`. Es obligatorio otorgar acceso a la herramienta `"Agent"` al orquestador para que la delegación sea posible.

PYTHON · SUB-AGENTES

```python
from claude_agent_sdk import AgentDefinition, ClaudeAgentOptions

options = ClaudeAgentOptions(
    allowed_tools=["Agent"],  # Habilita la capacidad de delegar
    agents={
        "data_auditor": AgentDefinition(
            description="Revisa tablas masivas y extrae anomalías estadísticas.",
            prompt="Eres un auditor de datos riguroso. Identifica outliers.",
            tools=["Read", "Grep"],  # Restringido a operaciones de lectura
            model="claude-haiku-4-5"  # Modelo de alta velocidad
        ),
        "code_reviewer": AgentDefinition(
            description="Analiza repositorios Python buscando vulnerabilidades.",
            prompt="Eres un ingeniero de seguridad. Previene inyecciones SQL.",
            tools=["Read", "Grep", "Glob"],
            model="claude-sonnet-4-6"  # Razonamiento profundo
        )
    }
)
```

Gestión de Permisos y Hooks

Los permisos granulares y la gobernanza del agente se controlan a través de los **Hooks**. Estos son callbacks que el sistema invoca durante eventos del ciclo de vida, permitiendo interceptar, mutar o denegar operaciones de manera determinista **antes** de que el modelo tenga oportunidad de fallar.

LOS 10 EVENTOS DE HOOKS DEL SDK PYTHON

El SDK Python soporta exactamente diez eventos. El PDF de respaldo solo menciona dos (`PreToolUse` y `PostToolUse`), perdiendo capacidades de gobernanza importantes:

EVENTO	CUÁNDO DISPARA
`PreToolUse`	Antes de ejecutar una herramienta. Más crítico para gobernanza — puede denegar/mutar.
`PostToolUse`	Después de ejecutar exitosamente. Para auditoría o inyección de contexto adicional.
`PostToolUseFailure`	Cuando una herramienta falla. Para logging de errores o reintentos.
`UserPromptSubmit`	Cuando el usuario envía un prompt. Para sanitización o inyección de contexto inicial.
`Stop`	Cuando el agente principal se detiene.
`SubagentStop`	Cuando un sub-agente se detiene. Incluye `agent_id` y `agent_transcript_path`.
`SubagentStart`	Cuando un sub-agente arranca. Para tracking del fan-out.
`PreCompact`	Antes de comprimir el contexto. Tiene `trigger: "manual" | "auto"`.
`Notification`	Para eventos de notificación del sistema.
`PermissionRequest`	Cuando se necesita una decisión de permisos. Permite manejar aprobaciones programáticamente.

Nota: El TypeScript SDK tiene 8 eventos adicionales (`SessionStart`, `SessionEnd`, `Setup`, `TeammateIdle`, `TaskCompleted`, `ConfigChange`, `WorktreeCreate`, `WorktreeRemove`) que **no están disponibles en Python aún.**

El evento más crítico es `PreToolUse`. Su firma exige recibir la información de entrada, el identificador y el contexto. En el siguiente ejemplo, un hook audita las ejecuciones de Bash para prevenir la alteración de directorios críticos.

PYTHON · HOOK DE SEGURIDAD

```python
from typing import Dict, Any
from claude_agent_sdk import HookMatcher, HookContext

async def enforce_safe_bash(
    input_data: Dict[str, Any],
    tool_id: str | None,
    context: HookContext
) -> Dict[str, Any]:
    tool_input = input_data.get("tool_input", {})
    command = str(tool_input.get("command", "")).lower()

    # Evaluación determinista de seguridad
    if "rm " in command or "chmod" in command:
        return {
            "hookSpecificOutput": {
                "hookEventName": "PreToolUse",
                "permissionDecision": "deny",
                "permissionDecisionReason":
                    "Comandos destructivos de Bash están restringidos."
            }
        }
    return {}  # diccionario vacío = la operación procede

hooks_config = {
    "PreToolUse": [HookMatcher(matcher="Bash", hooks=[enforce_safe_bash])]
}
```

Sesiones: capturar, reanudar, bifurcar

El SDK persiste automáticamente el historial de cada sesión en disco bajo `~/.claude/projects/<encoded-cwd>/<session-id>.jsonl`. `<encoded-cwd>` es el directorio de trabajo absoluto con cada caracter no-alfanumérico reemplazado por `-` (ej. `/Users/usuario/proj` → `-Users-usuario-proj`). Tres formas de retomar:

MODO	CÓMO FUNCIONA
`continue_conversation=True`	Reanuda la sesión más reciente del directorio actual. **No necesita ID**. Ideal cuando tu app corre una conversación a la vez.
`resume=session_id`	Reanuda una sesión específica por ID. Necesita capturar el ID previamente. Única opción con múltiples sesiones simultáneas (multi-tenant).
`resume=...` + `fork_session=True`	**Bifurca** creando una sesión nueva que parte con copia del historial original. La original queda intacta. Para explorar alternativas sin perder el thread principal.

CAPTURAR EL `SESSION_ID`

El ID está disponible en dos lugares:

- En el **`SystemMessage`** **init** emitido al inicio: en Python está anidado en `SystemMessage.data["session_id"]`.
- En el **`ResultMessage`** al final: campo directo `message.session_id`, presente siempre (éxito o error).

PYTHON · CAPTURAR + REANUDAR + BIFURCAR

```python
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions, ResultMessage

async class="tok-kw">def class="tok-fn">main():
    session_id = None

    # 1. Primera invocación: captura session_id
    async for msg in query(
        prompt="Analiza el módulo de autenticación",
        options=ClaudeAgentOptions(allowed_tools=["Read", "Grep"])
    ):
        if isinstance(msg, ResultMessage):
            session_id = msg.session_id
            print(f"Sesión original: {session_id}")

    # 2. Reanudar: el agente ya tiene contexto previo
    async for msg in query(
        prompt="Ahora implementa la refactorización sugerida",
        options=ClaudeAgentOptions(
            resume=session_id,
            allowed_tools=["Read", "Edit", "Write"]
        )
    ):
        if isinstance(msg, ResultMessage):
            print(msg.result)

    # 3. Bifurcar: explora alternativa sin contaminar la original
    forked_id = None
    async for msg in query(
        prompt="En vez de JWT, implementa OAuth2",
        options=ClaudeAgentOptions(
            resume=session_id,
            fork_session=True   # ← crea NUEVO session_id
        )
    ):
        if isinstance(msg, ResultMessage):
            forked_id = msg.session_id
            print(f"Sesión bifurcada: {forked_id}")

    # Ahora tienes DOS session_ids apuntando a dos historiales
    # distintos que puedes reanudar por separado.

asyncio.run(main())
```

CUIDADOS CON SESIONES

(1) Las sesiones persisten **conversación**, no el filesystem. Para revertir cambios de archivos usa *file checkpointing*. (2) El `cwd` debe coincidir entre la invocación original y la de retoma — el SDK busca el .jsonl bajo el directorio codificado. (3) Para reanudar entre máquinas (CI, contenedores efímeros), debes mover el archivo .jsonl o usar un `SessionStore` adapter.

Trazabilidad de Sub-agentes: `parent_tool_use_id`

Cuando un sub-agente genera mensajes (via la herramienta `Agent`), éstos llevan el campo `parent_tool_use_id` con el ID de la invocación que los originó. Esto permite reconstruir el árbol de delegación para auditoría:

PYTHON · TRAZAR SUB-AGENTES

```python
async for msg in query(prompt=prompt, options=options):
    if isinstance(msg, AssistantMessage):
        if msg.parent_tool_use_id:
            # Este mensaje vino de un sub-agente
            print(f"[sub-agent {msg.parent_tool_use_id}]: {msg.content}")
        else:
            # Mensaje del orquestador principal
            print(f"[orchestrator]: {msg.content}")
```

Parsing de mensajes y debugging

El motor devuelve un flujo asíncrono de tipos estrictos. Comprender la estructura de `AssistantMessage` (turnos intermedios) y `ResultMessage` (finalización) es vital para el registro de auditoría (*logging*).

PYTHON · DEBUGGING

```python
import asyncio
from claude_agent_sdk import (
    query, ClaudeAgentOptions,
    AssistantMessage, ResultMessage, TextBlock
)

async def debuggable_agent() -> None:
    options = ClaudeAgentOptions(allowed_tools=["Bash"], max_turns=5)

    async for msg in query(prompt="Lista los procesos activos.", options=options):
        # Tracking de eventos intermedios
        if isinstance(msg, AssistantMessage):
            for block in msg.content:
                if isinstance(block, TextBlock):
                    print(f"[texto]: {block.text}")
                elif block.type == "tool_use":
                    print(f"[tool]: {block.name} args={block.input}")
        # Consolidación final
        elif isinstance(msg, ResultMessage):
            if msg.subtype == "error_max_turns":
                print("⚠ El agente entró en bucle infinito y fue detenido.")
            elif msg.subtype == "success":
                print(f"✓ Respuesta final: {msg.result}")
                print(f"  Costo total: ${msg.total_cost_usd}, Tokens: {msg.total_tokens}")

if __name__ == "__main__":
    asyncio.run(debuggable_agent())
```

PRESCRIPCIÓN TÉCNICA

Para depuración avanzada, siempre captura y registra el estado de `msg.subtype` en los `ResultMessage`. Si tu agente falla frecuentemente por límite de turnos (`error_max_turns`), significa que **carece de las herramientas necesarias para observar el resultado de sus acciones**, o el prompt del sistema es ambiguo frente al formato de éxito esperado.

04 PARTE 04

Patrones de diseño con Agent SDK.

El verdadero potencial de la arquitectura agentizada se materializa al orquestar los componentes en patrones probados en producción. A continuación se detallan implementaciones completas para resolver problemas complejos de ingeniería de datos y automatización.

Pattern A

Agente Investigador

Recolecta información en tiempo real con WebSearch + WebFetch. Mantenimiento de contexto multi-página.

Pattern B

Clasificador con Hooks

Sistemas de alta precisión: hook fuerza investigación adicional si la confianza es baja.

Pattern C

Orquestador con Sub-agentes

Tareas amplias divididas entre especialistas para evitar saturación del contexto principal.

Pattern D

File System Batch

Worker de ETL sobre carpetas locales. Glob + Read + Write + Bash en sandbox.

Pattern E

Loop Autónomo Cron

Cron/daemon con `max_turns` y `max_budget_usd` como guardrails inquebrantables.

Pattern A · Agente Investigador

Caso de uso: Necesitas recolectar información en tiempo real sobre entidades dinámicas (ej. fusiones corporativas, regulaciones gubernamentales (ejemplo)) sintetizando datos desde internet, filtrando ruido y emitiendo un documento citado. **Ventaja Agent SDK:** Automatiza la ejecución de búsquedas, descarga del DOM limpio (WebFetch) y el mantenimiento del contexto a través de múltiples páginas leídas.

PYTHON · PATRÓN A

```python
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions, ResultMessage

async def investigative_pattern(target_entity: str) -> None:
    options = ClaudeAgentOptions(
        system_prompt="""
        Eres un investigador financiero senior. Sigue este flujo estricto:
        1. Utiliza WebSearch para buscar noticias recientes sobre el objetivo.
        2. Utiliza WebFetch para leer el contenido completo de al menos 3 fuentes.
        3. Sintetiza la información enfocándote en riesgo financiero.
        4. Emite el reporte final en formato Markdown, citando obligatoriamente las URLs.
        """,
        allowed_tools=["WebSearch", "WebFetch"],
        model="claude-sonnet-4-6",
        max_turns=10  # Previene exploración infinita
    )

    prompt = f"Realiza una investigación de riesgo sobre {target_entity} en los últimos 3 meses."
    print(f"Iniciando investigación autónoma sobre: {target_entity}...")

    async for msg in query(prompt=prompt, options=options):
        if isinstance(msg, ResultMessage):
            if msg.subtype == "success":
                print("\n═══ REPORTE DE INVESTIGACIÓN ═══")
                print(msg.result)
            else:
                print(f"\n[Fallo en la investigación]: {msg.subtype}")

if __name__ == "__main__":
    asyncio.run(investigative_pattern("Empresa Minera A (ejemplo)"))
```

ERROR COMÚN A EVITAR

No limitar los turnos (`max_turns`). Un agente de investigación sin límite puede entrar en un bucle recursivo de paginación web, consumiendo tokens de manera desmedida.

Pattern B · Clasificador con Verificación (Hooks)

Caso de uso: Sistemas de categorización donde la precisión es crítica (ej. transacciones bancarias dudosas). El modelo debe intentar clasificar, pero se ve forzado por un Hook de validación a buscar contexto adicional si su confianza es baja, garantizando que no se modifique el sistema de registro sin evidencia sólida.

PYTHON · PATRÓN B CON QUERY()

```python
import asyncio
from typing import Dict, Any
from claude_agent_sdk import (
    query, ClaudeAgentOptions, HookMatcher, HookContext,
    ResultMessage, tool, create_sdk_mcp_server
)

# Hook que evalúa el payload ANTES de escribir a la base de datos
async def verify_classification_confidence(
    input_data: Dict[str, Any], tool_id: str | None, context: HookContext
) -> Dict[str, Any]:
    tool_input = input_data.get("tool_input", {})
    confidence = float(tool_input.get("confidence_score", 0.0))

    if confidence < 0.8:
        return {
            "systemMessage": (
                f"Clasificación rechazada por baja confianza ({confidence}). "
                "Usa WebSearch para investigar el comercio antes de reintentar."
            ),
            "hookSpecificOutput": {
                "hookEventName": "PreToolUse",
                "permissionDecision": "deny",
                "permissionDecisionReason": "Confianza insuficiente para escritura en BD."
            }
        }
    return {}

@tool("commit_classification", "Guarda la categoría final",
      {"category": str, "confidence_score": float})
async def commit_classification(args: Dict[str, Any]) -> Dict[str,
Any]:
    return {"content": [{"type": "text", "text": "OK guardado"}]}

db_mcp = create_sdk_mcp_server("db_tools", "1.0.0",
                               tools=[commit_classification])

async def classifier_pattern(transaction_desc: str) -> None:
    options = ClaudeAgentOptions(
        system_prompt="""
        Eres un clasificador de gastos bancarios. Si dudas del rubro
        del comercio, debes investigarlo. Guarda la clasificación
        indicando tu nivel de confianza (0.0 a 1.0).
        """,
        mcp_servers={"database": db_mcp},
        allowed_tools=[
            "mcp__db_tools__commit_classification",
            "WebSearch"
        ],
        hooks={
            "PreToolUse": [HookMatcher(
                matcher="mcp__db_tools__commit_classification",
                hooks=[verify_classification_confidence]
            )]
        }
    )

    async for msg in query(prompt=f"Clasifica: {transaction_desc}",
                           options=options):
```

```
        if isinstance(msg, ResultMessage):
            print(msg.result)

if __name__ == "__main__":
    asyncio.run(classifier_pattern("Cobro de PROCESADOR-PAGOS-B"))
```

ERROR COMÚN A EVITAR

Devolver errores como simples cadenas de texto. El uso de `systemMessage` dentro del output del hook es fundamental para guiar al modelo sobre **cómo corregir el fallo**.

CUÁNDO MIGRAR A CLAUDESDKCLIENT (PATRÓN B)

La versión con `query()` arriba es técnicamente correcta y suficiente para la mayoría de casos. Migra a `ClaudeSDKClient` cuando necesites: (1) **interrumpir mid-task** si el hook detecta peligro crítico (vía `await client.interrupt()`), o (2) **cambiar `permission_mode` dinámicamente** durante la clasificación (vía `await client.set_permission_mode("dontAsk")`) cuando la confianza se desploma.

PYTHON · PATRÓN B CON CLAUDESDKCLIENT

```python
import asyncio
from claude_agent_sdk import (
    ClaudeSDKClient, ClaudeAgentOptions, HookMatcher,
    ResultMessage, tool, create_sdk_mcp_server
)

# Mismo hook y herramientas que arriba (verify_confidence, commit_classification)
# ... omitido por brevedad

async def classifier_pattern_client(transactions: list[str]) -> None:
    options = ClaudeAgentOptions(
        system_prompt="Clasificador de gastos. Si dudas, investiga.",
        mcp_servers={"database": db_mcp},
        allowed_tools=["mcp__db_tools__commit_classification", "WebSearch"],
        hooks={
            "PreToolUse": [HookMatcher(
                matcher="mcp__db_tools__commit_classification",
                hooks=[verify_classification_confidence]
            )]
        }
    )

    # Una sola conexión, múltiples clasificaciones en la misma sesión
    async with ClaudeSDKClient(options=options) as client:
        for tx in transactions:
            await client.query(f"Clasifica: {tx}")
            async for msg in client.receive_response():
                if isinstance(msg, ResultMessage):
                    print(msg.result)

            # Si en alguna iteración decido endurecer permisos:
            if some_risk_signal_detected():
                await client.set_permission_mode("dontAsk")

if __name__ == "__main__":
    asyncio.run(classifier_pattern_client([
        "Cobro de PROCESADOR-PAGOS-B",
        "Transferencia a ID tributario 00.000.000-0",
    ]))
```

Pattern C · Orquestador con Sub-agentes

Caso de uso: Tareas analíticas amplias donde un solo modelo saturaría su contexto (ej. un pipeline de scoring inmobiliario que requiere escrutinio geográfico, análisis financiero y resumen ejecutivo). El orquestador divide el problema.

PYTHON · PATRÓN C CON QUERY()

```python
import asyncio
from claude_agent_sdk import (
    query, ClaudeAgentOptions, AgentDefinition, ResultMessage
)

async def orchestrator_pattern(property_details: str) -> None:
    options = ClaudeAgentOptions(
        allowed_tools=["Agent"],  # Habilitación de la delegación
        agents={
            "geo-analyst": AgentDefinition(
                description="Especialista en demografía y plusvalía urbana.",
                prompt="Investiga el sector con WebSearch. Determina cercanía a servicios.",
                tools=["WebSearch", "WebFetch"],
                model="claude-haiku-4-5"  # Rápido y económico
            ),
            "financial-modeler": AgentDefinition(
                description="Calcula métricas de rentabilidad (Cap Rate).",
                prompt="Dadas las variables financieras y el informe geográfico, "
                       "calcula el Opportunity Score del 1 al 100. No busques en "
                       "internet, solo calcula.",
                tools=["Read"],
                model="claude-sonnet-4-6"  # Razonamiento profundo
            )
        }
    )

    prompt = f"""
    Evalúa esta propiedad para inversión: {property_details}.
    Plan de acción:
    1. Llama al agente 'geo-analyst' para análisis del barrio.
    2. Pasa el reporte geográfico y los datos de precio al 'financial-modeler'.
    3. Entrega el veredicto consolidado al usuario.
    """

    async for msg in query(prompt=prompt, options=options):
        if isinstance(msg, ResultMessage):
            print(msg.result)

if __name__ == "__main__":
    asyncio.run(orchestrator_pattern(
        "Depto 2D2B en un barrio céntrico (ejemplo), 3500 UI"
    ))
```

ERROR COMÚN A EVITAR

Asignar la herramienta `"Agent"` a los propios sub-agentes. Los sub-agentes no deben generar recursión de orquestación: esto fragmenta la trazabilidad y multiplica los costos.

CUÁNDO MIGRAR A CLAUDESDKCLIENT (PATRÓN C)

La versión con `query()` arriba funciona para una orquestación one-shot. Migra a `ClaudeSDKClient` cuando necesites **preservar contexto entre invocaciones sucesivas al orquestador** (no dentro de una sola query, sino llamadas separadas — ej. un dashboard que pregunta al mismo orquestador secuencialmente sobre múltiples propiedades). Con `ClaudeSDKClient`, los hallazgos de la propiedad #1 quedan en contexto para evaluar la #2 sin re-ensamblar.

PYTHON · PATRÓN C CON CLAUDESDKCLIENT

```python
import asyncio
from claude_agent_sdk import (
    ClaudeSDKClient, ClaudeAgentOptions,
    AgentDefinition, ResultMessage
)

async class="tok-kw">def class="tok-fn">evaluate_portfolio(properties: list[str]) → None:
    options = ClaudeAgentOptions(
        allowed_tools=["Agent"],
        agents={
            "geo-analyst": AgentDefinition(
                description="Demografía y plusvalía urbana.",
                prompt="Investiga el sector con WebSearch.",
                tools=["WebSearch", "WebFetch"],
                model="haiku"
            ),
            "financial-modeler": AgentDefinition(
                description="Calcula Opportunity Score.",
                prompt="Combina geo + precio. Dame score 1-100.",
                tools=["Read"],
                model="sonnet"
            )
        }
    )

    # Una sola sesión preserva contexto entre propiedades
    async with ClaudeSDKClient(options=options) as client:
        for prop in properties:
            await client.query(
                f"Evalúa {prop}. Compara con las propiedades anteriores "
                f"de esta sesión si las hay."
            )
            async for msg in client.receive_response():
                if isinstance(msg, ResultMessage):
                    print(f"\n═══ {prop} ═══\n{msg.result}")

if __name__ == "__main__":
    asyncio.run(evaluate_portfolio([
        "Depto 2D2B en un barrio céntrico (ejemplo), 3500 UI",
        "Casa en un barrio residencial (ejemplo), 8000 UI",
        "Studio en el centro (ejemplo), 1800 UI",
    ]))
```

Pattern D · Agente con File System (Batch)

Caso de uso: Integración de sistemas heredados donde los datos fluyen a través de carpetas locales (archivos de logs, facturas parseadas desde correos). El agente actúa como un worker de ETL.

PYTHON · PATRÓN D

```python
import asyncio
import os
from claude_agent_sdk import query, ClaudeAgentOptions, ResultMessage

async class="tok-kw">def class="tok-fn">filesystem_agent_pattern() -> None:
    # Preparación del entorno de pruebas
    work_dir = "/tmp/banking_batch"
    os.makedirs(os.path.join(work_dir, "raw_data"), exist_ok=True)
    os.makedirs(os.path.join(work_dir, "processed"), exist_ok=True)
    with open(os.path.join(work_dir, "raw_data", "batch_001.txt"), "w") as f:
        f.write("TXN: 15/05/2026 - COMERCIO-D - $55000\n"
                "TXN: 16/05/2026 - SERVICIOS-C - $32000")

    options = ClaudeAgentOptions(
        system_prompt="""
        Eres un worker de procesamiento de datos por lotes (batch processing).
        1. Utiliza Glob para encontrar archivos .txt en ./raw_data/
        2. Utiliza Read para procesar el contenido de cada archivo.
        3. Convierte las transacciones a formato JSON.
        4. Utiliza Write para guardar el resultado en ./processed/ con sufijo _done.json.
        5. Una vez escrito, borra el archivo original usando Bash para evitar reprocesamiento.
        """,
        allowed_tools=["Glob", "Read", "Write", "Bash"],
        permission_mode="acceptEdits",  # Ejecución silenciosa
        cwd=work_dir  # Sandbox restringido a este directorio
    )

    async for msg in query(prompt="Inicia la rutina de procesamiento de archivos.",
                           options=options):
        if isinstance(msg, ResultMessage):
            print(f"Rutina finalizada con éxito. Detalle: {msg.result}")

if __name__ == "__main__":
    asyncio.run(filesystem_agent_pattern())
```

ERROR COMÚN A EVITAR

Operar sin restringir el parámetro `cwd` (Current Working Directory). Sin un entorno sandbox, un error en la interpretación del modelo utilizando la herramienta Bash podría alterar la estructura principal del sistema anfitrión.

Pattern E · Loop Autónomo (Operaciones Cron/Daemon)

Caso de uso: Monitoreo y auto-corrección sin supervisión humana. Ideal para auditar trabajos del día, notificar anomalías y mantener trazabilidad de los gastos asíncronos.

PYTHON · PATRÓN E

```python
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions, ResultMessage

async def autonomous_cron_pattern() -> None:
    # Este script está diseñado para ser gatillado por cron u Airflow
    options = ClaudeAgentOptions(
        system_prompt="""
        Eres un auditor de la base de datos de transacciones.
        Revisa las transacciones usando la herramienta Bash con scripts locales.
        Si encuentras inconsistencias en categorización, aplica la corrección.
        Maneja tu propio flujo y finaliza indicando "AUDITORÍA COMPLETA".
        """,
        allowed_tools=["Bash", "Read", "Write"],
        permission_mode="acceptEdits",
        # Guardrails vitales para procesos desatendidos
        max_turns=20,
        max_budget_usd=2.0
    )

    try:
        async for msg in query(prompt="Inicia la auditoría nocturna de integridad.",
                               options=options):
            if isinstance(msg, ResultMessage):
                if msg.subtype == "error_max_budget_usd":
                    print("ALERTA: El proceso excedió el límite financiero asignado. "
                          "Intervención manual requerida.")
                elif msg.subtype == "error_max_turns":
                    print("ALERTA: El agente entró en bucle infinito. Turnos agotados.")
                elif msg.subtype == "success":
                    print("EJECUCIÓN EXITOSA:")
                    print(msg.result)
                    print(f"Costos: ${msg.total_cost_usd} / Tokens: {msg.total_tokens}")
    except Exception as e:
        print(f"Fallo catastrófico en el runtime del agente: {e}")

if __name__ == "__main__":
    asyncio.run(autonomous_cron_pattern())
```

ERROR COMÚN A EVITAR

Desplegar agentes autónomos (cron) sin establecer `max_turns` o `max_budget_usd`. Un fallo en el parseo de una herramienta podría inducir al agente a intentar solventar el error eternamente, consumiendo la cuota mensual de tokens en pocas horas durante la madrugada.

POR QUÉ NO MIGRAR PATRÓN E A CLAUDESDKCLIENT

Para procesos cron/daemon, `query()` es la elección correcta. Cada ejecución de cron es independiente: no hay continuidad de sesión que preservar, no hay interrupciones interactivas que manejar, no hay cambio de modo mid-task. El overhead de `ClaudeSDKClient` (manejo manual de connect/disconnect, gestión de buffers) no aporta nada en este escenario.

05 PARTE 05

Costos y performance.

La introducción del entorno de ejecución agentizado requiere recalibrar el modelo de costos y comprender las concesiones entre latencia e infraestructura.

Facturación y el Crédito Mensual del Agent SDK (Junio 2026)

Históricamente, el uso de herramientas asíncronas de línea de comandos drenaba los presupuestos generosos pero finitos que Anthropic destinaba a las interfaces de usuario web interactivo. Para solventar este desajuste en el consumo de cómputo no-interactivo, Anthropic introduce un nuevo paradigma efectivo a partir del **15 de junio de 2026**.

Disponibilidad: Planes Pro, Max, Team y Enterprise. **NO incluye** cuentas Claude Developer Platform usando API key tradicional (éstas mantienen pay-as-you-go).

CRÉDITO MENSUAL POR PLAN (OFICIAL)

PLAN	CRÉDITO MENSUAL
Pro	$20
Max 5×	$100
Max 20×	$200
Team — Standard seats	$20
Team — Premium seats	$100
Enterprise (usage-based)	$20
Enterprise (seat-based, Premium seats)	$200
*⚠ Miembros de planes Enterprise seat-based en **Standard seats no son elegibles** para reclamar el crédito.*	

QUÉ CUBRE EL CRÉDITO

- Claude Agent SDK en proyectos propios (Python o TypeScript)
- El comando `claude -p` en Claude Code (modo no-interactivo)
- La integración Claude Code GitHub Actions

- Apps de terceros que autentican con tu suscripción Claude vía Agent SDK

QUÉ NO CUBRE

- Claude Code interactivo en terminal o IDE
- Conversaciones Claude en web, desktop o mobile
- Claude Cowork
- Otras features que consumen *extra usage*

CÓMO FUNCIONA EL CRÉDITO — CINCO REGLAS

1. **Per-user, no pooleable.** Los créditos pertenecen a cuentas individuales. No se pueden compartir, poolear ni transferir entre miembros del equipo.
2. **Refresca mensualmente.** Se resetea al inicio de cada ciclo de facturación. **Los créditos no usados NO se acumulan** al siguiente ciclo.
3. **Opt-in una sola vez.** Reclamas el crédito a través de tu cuenta Claude una vez; después refresca automáticamente cada ciclo.
4. **Drains first.** El uso del Agent SDK consume del crédito mensual antes que cualquier otra fuente.
5. **Más allá del crédito → extra usage.** Cuando agotas el crédito mensual, el uso adicional pasa a *extra usage* a tarifas API estándar — pero solo si tienes extra usage habilitado. **Si no está habilitado, los requests del Agent SDK se detienen** hasta el siguiente refresh.

> **PARA ADMINISTRADORES TEAM/ENTERPRISE**
>
> Los créditos son **por-usuario, no pooleables** entre la organización. Cada usuario elegible reclama el suyo. El crédito mensual está dimensionado para experimentación y automatización individual. **Los equipos que corren automatización de producción compartida deben usar Claude Developer Platform con API key para billing pay-as-you-go predecible.**

AUTENTICACIÓN: CÓMO ACTIVAR EL CRÉDITO EN TU CÓDIGO

El crédito subsidiado **solo aplica al Agent SDK autenticado vía OAuth de usuario.** Si usas API keys estándar (`sk-ant-...`) dentro del Agent SDK, el tráfico recae sobre pay-as-you-go normal. Para inyectar tu OAuth token:

SHELL + PYTHON · ACTIVAR CRÉDITO

```
# Paso 1: genera el token en tu terminal (una vez)
# $ claude setup-token
# → Te entrega un CLAUDE_CODE_OAUTH_TOKEN

# Paso 2: inyecta como variable de entorno en CI/CD o tu script
# $ export CLAUDE_CODE_OAUTH_TOKEN=P0P

# Paso 3: usa Agent SDK normalmente; el SDK detecta el token automáticamente
import asyncio
from claude_agent_sdk import query, ClaudeAgentOptions

async class="tok-kw">def class="tok-fn">main():
    async for msg in query(
        prompt="Análisis del ticker AAPL",
        options=ClaudeAgentOptions(model="claude-sonnet-4-6")
    ):
        print(msg)

asyncio.run(main())
# Este request consume del crédito mensual de tu plan Max/Pro/Team/Enterprise
```

Comparación de latencia y performance (TTFT)

La comodidad orquestal del Agent SDK tiene un precio en velocidad. El **Tiempo hasta el Primer Token** (TTFT, por sus siglas en inglés) es la métrica de fricción principal al comparar enfoques.

Estudios de *benchmarking* en la arquitectura del agente muestran que un flujo ejecutado mediante el **SDK Estándar** (llamadas sin estado con herramientas inyectadas) inicia la transmisión en aproximadamente **~20 milisegundos**, completando resoluciones de texto simples en **~2.5 segundos**.

Por el contrario, el **Claude Agent SDK**, al invocar herramientas a través del servidor MCP integrado en Python y resolver el empaquetado del entorno de red, incrementa el TTFT promedio a **~2.86 segundos**, llevando el tiempo total por turno a rangos de **~8.3 segundos** para tareas combinadas con Bash o sistema de archivos.

ARQUITECTURA	TTFT	TURNO SIMPLE	TURNO CON BASH/FS	RECOMENDADO PARA
SDK Estándar				
~20 ms				
~2.5 s				
N/A				

Bots de mensajería, UIs en tiempo real	
Agent SDK	
~2.86 s	
~3 s	
~8.3 s	
Background workers, batch, cron	

ANÁLISIS DE VELOCIDAD

Si la arquitectura de tu sistema recae en el diseño de un bot de mensajería (ej. WhatsApp) donde el retraso del usuario degrada la experiencia de manera inaceptable, **el SDK Estándar es imperativo.** El motor Agent SDK fue concebido para trabajos asíncronos de fondo (*background workers*), procesamiento por lotes, o rutinas complejas donde esperar 30 segundos por una orquestación precisa de cinco herramientas justifica la demora.

Optimización: `effort` y caching

Para controlar el consumo de tokens en tareas extensas, el parámetro `effort` altera la profundidad del razonamiento adaptativo del modelo. Los valores oficiales actuales son cuatro:

- `"low"` — Reduce dramáticamente la latencia y los costos de salida. Es imperativo para sub-agentes encargados de tareas mecánicas y poco reflexivas, como buscar archivos en un directorio usando Glob o compilar resúmenes de datos puramente estructurales.
- `"medium"` — Balance estándar por defecto.
- `"high"` — Indispensable para refactorización matemática de datos complejos (como analítica financiera o modelado).
- `"max"` — Máximo razonamiento. *(Nota: el PDF de respaldo decía `"xhigh"`, valor que **no existe** en la API.)*

RIESGO DE CACHE INVALIDADO

El control de costos recae en el uso magistral del **Prompt Caching** de Anthropic. El caché se calcula del lado del servidor tomando el prefijo exacto de los tokens del sistema y el contexto. Modificar dinámicamente el valor del parámetro `effort` a través del código en turnos sucesivos o en hilos de sub-agentes anidados que comparten un prefijo altera la configuración de salida. Esta mutación forzará un *Cache Miss* (invalidación del caché), obligando al sistema a re-procesar (y cobrar) la totalidad de los archivos de contexto cargados, volatilizando miles de tokens en cada turno inútilmente. **Qué hacer:** Establece un perfil de `effort` fijo en tu `ClaudeAgentOptions` base y diséñelo alrededor del requisito general del trabajador.

06 PARTE 06

Casos donde NO conviene migrar.

Es un riesgo fundamental de la ingeniería de software adoptar una nueva arquitectura asumiendo que reemplaza a la anterior de forma omnipotente. El Agent SDK es un marco de automatización asíncrona; migrar código operativo del SDK Estándar hacia este paradigma puede representar *over-engineering* grave en escenarios específicos.

Escenarios de over-engineering identificados

1

Clasificadores simples (Input → Output)

CASO Flujos estáticos donde el texto entra (ej. transacción transcribida desde email) y sale una etiqueta tipada (ej. `{"categoria": "supermercado"}`).

POR QUÉ DUELE El Agent SDK introduce tiempos de inicialización de entorno de casi 3 segundos y consume recursos gestionando el "bucle del agente" cuando no existe necesidad algorítmica de iterar.

QUÉ HACER Mantén estos flujos en el SDK Estándar utilizando el parámetro `response_format` para garantizar salidas JSON rápidas y predecibles.

2

Llamadas one-shot y transformaciones

CASO Scripts donde la invocación a Claude es meramente transformacional (traducción, resúmenes cortos, parseo de recibos a estructuras).

POR QUÉ DUELE Si la tarea se soluciona en un solo turno y carece de necesidad para navegar iterativamente el sistema de archivos local o buscar información dinámica en la web, integrar un Motor MCP y definir permisos de ejecución añade capas de complejidad innecesaria.

3

Ambientes de alta concurrencia (Multi-Tenant UIs)

CASO Tu arquitectura atiende peticiones en tiempo real de múltiples usuarios finales.

POR QUÉ DUELE El diseño del Agent SDK ata su ejecución al espacio de trabajo local (directorio `cwd`). Aislar las ejecuciones simultáneas de cientos de usuarios requeriría programar una envoltura de infraestructura capaz de levantar, mantener y destruir subprocesos, **bases de datos vectoriales aisladas y permisos Docker por cada petición**.

QUÉ HACER Para infraestructuras asíncronas de usuarios masivos, ecosistemas abstractos (como implementaciones nativas del SDK Estándar) garantizan menor fricción.

Heurísticas claras de decisión

Para diagnosticar la transición técnica, aplica estas métricas a la arquitectura de tu componente:

Heurística del Bucle de Control

Si tu código Python actual tiene que gestionar el flujo con declaraciones repetitivas como `while response.stop_reason == "tool_use":`, capturando variables, ejecutando funciones y re-empacando JSON de manera engorrosa — tu proyecto **"pide" migrar al Agent SDK**.

Heurística de Entropía Estructural

Si tu aplicación tiene como fin principal navegar más de un documento en disco, o necesita acceso robusto e interactivo para buscar en internet para completar una sola tarea — **la sobrecarga del Agent SDK está holgadamente justificada**.

Heurística de Latencia del Usuario

Si un ser humano interactivo se encuentra esperando una barra de progreso que requiere resolución en milisegundos (inferior a 2 segundos) — **el SDK Estándar sigue siendo lo correcto**.

07

PARTE 07

Aplicación a mis proyectos.

Evaluación técnica para los proyectos del ecosistema de finanzas analíticas, justificando el marco arquitectónico apropiado.

1 · FinTrack

SDK Estándar

Clasificador de transacciones parseadas.

Este flujo corresponde a un clasificador determinista **Input → Output**. Dado que el parsing desde los correos ya está resuelto en Python, inyectar el texto para obtener un string que diga "Gastos Médicos" es una operación *one-shot*. Imponer el peso del motor del Agent SDK penalizaría la latencia (escalando el TTFT) y no extrae valor de la manipulación de sistemas de archivo o la ejecución de bucles.

2 · PropScan

Híbrido

Scraping y Opportunity Score.

Para la extracción de datos inicial de portales como un portal inmobiliario público, el raspado (*scraping*) determinista vía **Scrapy**/**Playwright** operado en Python crudo sigue siendo insustituible por su previsibilidad; los LLM procesando árboles HTML crudos despilfarran un volumen excesivo de tokens.

Para la fase de análisis ("Scoring" y "Contextualización"), migrar a Agent SDK aporta un salto de valor. Implementando el **Patrón C** (**Orquestador**), un sub-agente especializado (`claude-haiku-4-5`) puede utilizar WebSearch para investigar desarrollos de infraestructura en el barrio de cada departamento. Posteriormente, el orquestador principal (`claude-sonnet-4-6`) correlaciona los hallazgos geográficos con las tablas de precios recolectadas y genera el Opportunity Score.

3 · StockScope

Agent SDK

Evaluación de tickers.

La síntesis financiera es la aplicación definitiva para la automatización multi-turno. Implementando el **Patrón A** (**Investigador**), al proveer el ticker "TICKER-A" al agente, el motor operará en un bucle autónomo:

(1) Ejecutará un servidor MCP *in-process* que interactúe con la API de una API pública de finanzas u otra biblioteca Pandas para traer datos fundamentales; (2) Utilizará simultáneamente WebSearch para recuperar las tres noticias macroeconómicas más recientes del día asociadas al ticker; (3) Consolidará una perspectiva técnica y fundamental combinada, citando las URLs descubiertas de forma independiente. Todo el engorroso bucle es absorbido por la herramienta.

4 · Auditor de FinTrack

Agent SDK

Proyecto nocturno.

Representa el estado del arte de la automatización desatendida. Utilizando el **Patrón E** (**Loop Autónomo**) programado vía un Job Cronológico (*cron job*), el proceso no sufre presiones de latencia.

Configurándolo con `permission_mode="acceptEdits"`, el agente empleará la herramienta Read para auditar la base de datos o CSV de resultados del día. Si detecta transacciones ambiguas (ej. cobros exóticos de un procesador de pagos), invocará WebFetch para averiguar la naturaleza del cargo, y, usando herramientas MCP propias de modificación de BD, aplicará reclasificaciones correctivas directas. Un Hook `PostToolUse` puede enviar alertas asíncronas vía Slack/Telegram de cada modificación. Aprovechará directamente el crédito mensual de Claude Max.

5 · Bot de WhatsApp

SDK Estándar

FinTrack — proyecto futuro.

La experiencia del usuario es primordial en una plataforma de mensajería instantánea. Un retraso arquitectónico de base de casi 3 segundos provocado por la instanciación de red MCP del Agent SDK arruinaría la sensación de conversacionalidad.

El motor LLM debe operar con latencia mínima, reaccionando rápidamente a la consulta inyectando herramientas deterministas vía *function calling* convencional para la ejecución transaccional asíncrona sobre la base de datos y devolviendo texto inmediato.

08 PARTE 08 Roadmap de aprendizaje.

El siguiente plan de cuatro semanas despliega la transición técnica desde el conocimiento fundamental de la API de mensajes hacia la orquestación agentizada completa, integrando componentes incrementales a fin de evitar fricción de aprendizaje.

SEMANA 01

Adaptación arquitectónica

Internalizar el cambio de paradigma donde el motor asume el control del bucle de ejecución y manipular las primitivas nativas del sistema de archivos.

Ej. 01

Hello World Diagnóstico: instancia un agente con `query()` + `ClaudeAgentOptions`, accede a Bash, pídele reportar la versión de Python, imprime `message.result` y extrae `msg.total_cost_usd`.

Ej. 02

Manejador de archivos batch: implementa el Patrón D sin código a medida. Entrega al agente un directorio restringido (`cwd`) con tres archivos .txt desordenados. Usa Read, Glob y Write con `permission_mode="acceptEdits"` para leerlos, consolidar resumen y depositar archivo resultante.

SEMANA 02

Extensibilidad y gobernanza

Crear interfaces MCP nativas en Python y someter al motor a límites de seguridad inquebrantables.

Ej. 01

Servidor MCP Local: convierte un script existente que extrae métricas de una API o Pandas. Decóralo con `@tool` + type hints. Empaquétalo con `create_sdk_mcp_server` y registra en `mcp_servers`.

Ej. 02

Protección vía Hooks: configura un evento `PreToolUse` asociado a la **herramienta genérica Write**. Codifica la validación para impedir sobrescribir archivos que terminen en `.csv`. Prueba con un CSV intencional; observa el `deny` sin detener el hilo.

SEMANA 03

Orquestación y memoria

Optimizar tiempo y costos delegando componentes a modelos pequeños y controlando la sesión.

Ej. 01 **Patrón Orquestador:** instancia un orquestador con `"Agent"` en `allowed_tools`. Dos `AgentDefinition`: analista rápido con `claude-haiku-4-5` + WebSearch, y consolidador con `claude-sonnet-4-6`. Visualiza la delegación vía `AssistantMessage`.

Ej. 02 **Resumen Cíclico:** registra el `session_id` desde el `SystemMessage` de la primera transacción. Destruye el proceso, relanza con `resume=session_id` y pregunta si "recuerda" lo conversado. Comprueba la recarga desde los logs `.claude/projects` de forma invisible.

SEMANA 04

Despliegue en producción

Consolidar el conocimiento transicionando el primer proyecto operativo al entorno agentizado: **StockScope**.

Paso 1 Desarrolla dos herramientas `@tool`: una para extraer métricas con `yfinance`, otra para resumen Pandas. Agrupa en un servidor MCP in-process.

Paso 2 Instancia un orquestador con `claude-sonnet-4-6`, MCP local + WebSearch + WebFetch. Define `effort="high"` fijo.

Paso 3 Si tienes plan Claude Max, inyecta `CLAUDE_CODE_OAUTH_TOKEN` (generado con `claude setup-token`). Ejecuta el prompt por un ticker y observa cómo el agente ensambla el reporte económico con cero sobrecostos fuera de tu plan Max.

ANEXO DE VERIFICACIÓN

Verificación de completitud.

Cada ítem cruzado contra la documentación oficial de Anthropic. Los ítems originales del checklist del usuario están marcados ✓ si fueron verificados correctos, o anotados como **error original** si la doc oficial contradice el valor propuesto.

✓ **Los cinco valores de `permission_mode`**

Doc oficial Python (`PermissionMode` Literal): `default`, `acceptEdits`, `plan`, `dontAsk`, `bypassPermissions`. Documentados en Parte 2.

Nota: el PDF de respaldo dice `auto` — incorrecto, ese valor solo existe en TypeScript SDK.

✓ **Distinción `allowed_tools` (auto-aprueba) vs `disallowed_tools` (bloquea)**

Doc oficial Permissions: deny rules son checkeadas primero y overrideán incluso `bypassPermissions`. `allowed_tools` no restringe — solo pre-aprueba. Documentado en Parte 2 con la trampa de bypassPermissions.

✓ **Los diez eventos de hooks del SDK Python**

Doc oficial Python (`HookEvent` Literal): `PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `UserPromptSubmit`, `Stop`, `SubagentStop`, `PreCompact`, `Notification`, `SubagentStart`, `PermissionRequest`. Tabla completa en Parte 3.

Nota: TypeScript SDK tiene 8 eventos adicionales no disponibles aún en Python.

✓ **Default vacío de `system_prompt` y formato dict del preset `claude_code`**

Doc oficial Modifying System Prompts: default = None → SDK usa prompt mínimo (solo tool calling), distinto de `claude -p`. Formato preset: `{"type": "preset", "preset": "claude_code"}` con opcional `"append"`. Documentado en Parte 3.

✓ **El parámetro `setting_sources` y qué se rompe sin él**

Doc oficial Python: default = None → NO se carga ningún settings de filesystem. **Para cargar CLAUDE.md hay que incluir `"project"` explícitamente.** Precedencia local > project > user. Documentado en Parte 3.

! **Distinción entre `query()` y `ClaudeSDKClient`**

El checklist del usuario afirmaba: "`query()` no soporta tools custom ni hooks; `ClaudeSDKClient` sí los soporta". **Esto es incorrecto según la documentación oficial actual.**

Valor correcto según docs (`platform.claude.com/docs/en/agent-sdk/python`): ambas interfaces soportan ✓ **Hooks** y ✓ **Custom Tools**. La diferencia real es: continuidad de sesión, soporte de interrupciones (solo `ClaudeSDKClient` tiene `interrupt()`), y control explícito de lifecycle. Documentado con tabla comparativa en Parte 3 y patrones B y C muestran ambas versiones.

Nota: El README del repo en GitHub afirma lo opuesto — es una inconsistencia interna de la documentación de Anthropic. La SDK reference oficial es la fuente más actualizada.

✓ **`SystemMessage` init para capturar `session_id`**

Doc oficial Sessions: en Python el ID está anidado en `SystemMessage.data["session_id"]` al inicio. También disponible en `ResultMessage.session_id` al final (presente siempre, éxito o error). Ejemplo de código en Parte 3.

✓ **`parent_tool_use_id` para trazar mensajes de sub-agentes**

Doc oficial Python: campo presente en `UserMessage`, `AssistantMessage` y `StreamEvent`. Documentación en Parte 3 con snippet de ejemplo.

`fork_session` además de `resume`

Doc oficial Python (`ClaudeAgentOptions`): `fork_session: bool = False`. Combinado con `resume=session_id`, crea una sesión nueva con copia del historial. Tres modos de retoma completos (`continue_conversation`, `resume`, `fork_session`) documentados en Parte 3 con ejemplo completo.

Detalles del crédito Agent SDK Junio 2026

Los cuatro detalles del checklist del usuario (opt-in, per-user, no rollover, planes Team/Enterprise) están **todos confirmados** en la doc oficial. La doc oficial contiene **seis hechos adicionales** que el checklist no mencionaba y que sí están en v1.2:

- Tabla de créditos para Team Standard ($20), Team Premium ($100), Enterprise usage-based ($20), Enterprise seat-based Premium ($200)
- Enterprise seat-based en Standard seats NO son elegibles
- "Drains first": el crédito SDK se consume antes que cualquier otra fuente
- Excedido el crédito → extra usage **solo si está habilitado**; si no, los requests se detienen
- Cubre apps de terceros que autentican con tu suscripción vía Agent SDK
- Recomendación oficial para admins Team/Enterprise: producción a escala debe usar Claude Developer Platform con API key

Errores adicionales corregidos vs PDF de respaldo

CONCEPTO	PDF DE RESPALDO	VALOR OFICIAL CORREGIDO
`permission_mode` values	3 valores incluyendo `auto`	5 valores (`auto` es TypeScript-only)
`effort` literal	`"low" / "medium" / "high" / "xhigh"`	`"low" / "medium" / "high" / "max"`
Hooks events listados	Solo `PreToolUse` y `PostToolUse`	10 eventos completos en Python
Path de sesiones	`.claude/projects/`	`~/.claude/projects/<encoded-cwd>/<session-id>.jsonl`
Patrones B y C usan	Solo `query()`	Ambas versiones (`query()` + `ClaudeSDKClient`) con guía cuándo usar cuál
Distinción `allowed_tools` vs `disallowed_tools`	No mencionada	Documentada con trampa de `bypassPermissions`
Herencia de permisos en sub-agentes	No mencionada	Documentada (no puede overridearse — riesgo de seguridad)
Orden de evaluación de permisos	No mencionado	5 pasos documentados (Hooks → Deny → Mode → Allow → canUseTool)

G

GLOSARIO

Términos esenciales.

Vocabulario técnico para referencia rápida durante el estudio.

`Agent Loop`

Bucle de razonamiento del modelo: evaluar → llamar herramienta → recibir resultado → repetir. En el Agent SDK vive dentro de `queryLoop()`.

`queryLoop()`

Función asíncrona que comparten Claude Code y el Agent SDK como motor único de ejecución.

`MCP`

Model Context Protocol: protocolo abierto para conectar herramientas a LLMs. El Agent SDK soporta servidores MCP locales *in-process*, sin latencia de red. **Ver guía completa de MCP** para detalle del protocolo, primitivas, transports, OAuth 2.1 y portabilidad entre hosts.

`In-process MCP`

Servidor MCP que corre en el mismo proceso de Python. Se crea con `create_sdk_mcp_server`.

`ClaudeAgentOptions`

Configuración central del entorno del agente: modelo, herramientas, permisos, hooks, cwd, presupuestos.

`query()`

Interfaz funcional simple. Nueva sesión cada invocación. Soporta hooks y custom tools.

`ClaudeSDKClient`

Clase alternativa a `query()`. Mantiene una sesión entre múltiples `client.query()`. Soporta interrupciones (`interrupt()`) y cambios dinámicos de modo (`set_permission_mode()`) o modelo (`set_model()`).

`Hook`

Callback que el sistema invoca durante eventos del ciclo de vida. Permite interceptar, mutar o denegar.

`10 eventos hooks (Python)`

`PreToolUse`, `PostToolUse`, `PostToolUseFailure`, `UserPromptSubmit`, `Stop`, `SubagentStop`, `SubagentStart`, `PreCompact`, `Notification`, `PermissionRequest`.

`permission_mode`

Política de aprobación de herramientas. Cinco valores oficiales en Python: `default`, `acceptEdits`, `plan`, `dontAsk`, `bypassPermissions`. El valor `auto` existe solo en TypeScript SDK.

`allowed_tools`

Lista blanca de herramientas auto-aprobadas. **No restringe**: solo pre-aprueba. Para MCPs: `mcp__<servidor>__<tool>`.

`disallowed_tools`

Lista negra. Las herramientas listadas **siempre** se deniegan, incluso en modo `bypassPermissions`.

`setting_sources`

Lista de fuentes de settings a cargar (`"user"`, `"project"`, `"local"`). Default = None: NO carga settings ni CLAUDE.md. Para CLAUDE.md incluir `"project"`.

`system_prompt preset`

Dict `{"type": "preset", "preset": "claude_code"}` que activa el prompt completo Claude Code. Añade `"append": "..."` para extender. Sin esto, el SDK usa un prompt mínimo solo para tool calling.

`AgentDefinition`

Estructura que describe a un sub-agente: descripción, prompt, tools y modelo dedicado.

`TTFT`

Time To First Token. Latencia hasta el primer token emitido. SDK Estándar: ~20 ms · Agent SDK: ~2.86 s.

`effort`

Profundidad del razonamiento adaptativo: `low`, `medium`, `high`, `max`. **No** modificar entre turnos: invalida prompt cache.

`Prompt Caching`

Mecanismo de Anthropic que cachea prefijos. Cambiar `effort` dinámicamente invalida el caché y multiplica costos.

`session_id`

Identificador para restaurar sesiones desde `~/.claude/projects/` vía `resume=session_id`.

`continue_conversation`

Flag de `ClaudeAgentOptions`. Reanuda la sesión más reciente del directorio actual sin necesitar ID.

`fork_session` Flag de `ClaudeAgentOptions`. Con `resume=session_id` + `fork_session=True`, crea una nueva sesión con copia del historial original.
`ResultMessage` Mensaje final del flujo. Subtypes: `success`, `error_max_turns`, `error_max_budget_usd`.
`AssistantMessage` Turnos intermedios del agente con bloques `TextBlock` o `tool_use`.
`parent_tool_use_id` Campo en `UserMessage`, `AssistantMessage` y `StreamEvent`. Permite rastrear qué mensajes vienen de sub-agentes durante delegación.
`cwd` Current Working Directory: define el sandbox del agente. **Siempre restringirlo.**
`OAuth Token` `CLAUDE_CODE_OAUTH_TOKEN`: mecanismo para inyectar autenticación de usuario en CI/CD. Único modo de acceder al crédito mensual dedicado.
`max_turns / max_budget_usd` Guardrails para procesos desatendidos. Sin ellos un fallo de parseo puede agotar la cuota mensual.

R

REFERENCIAS

Fuentes oficiales.

Las 25 fuentes que sustentan este reporte. Las URLs de documentación oficial fueron verificadas contra `platform.claude.com/docs`, `code.claude.com/docs` y `support.claude.com`.

DOCUMENTACIÓN OFICIAL ANTHROPIC

Anthropic. *Agent SDK overview.* `code.claude.com/docs/en/agent-sdk/overview`

Anthropic. *Agent SDK reference — Python.* `code.claude.com/docs/en/agent-sdk/python`

Anthropic. *How the agent loop works.* `code.claude.com/docs/en/agent-sdk/agent-loop`

Anthropic. *Intercept and control agent behavior with hooks.* `code.claude.com/docs/en/agent-sdk/hooks`

Anthropic. *Subagents in the SDK.* `code.claude.com/docs/en/agent-sdk/subagents`

Anthropic. *Modifying system prompts.* `code.claude.com/docs/en/agent-sdk/modifying-system-prompts`

Anthropic. *Permissions.* `code.claude.com/docs/en/agent-sdk/permissions`

Anthropic. *Sessions.* `code.claude.com/docs/en/agent-sdk/sessions`

Anthropic. *Run prompts on a schedule.* `code.claude.com/docs/en/scheduled-tasks`

Anthropic. *Authentication.* `code.claude.com/docs/en/authentication`

Anthropic. *Use the Claude Agent SDK with your Claude plan.* `support.claude.com/en/articles/15036540`

Anthropic. *claude-agent-sdk-python — README.md.* `github.com/anthropics/claude-agent-sdk-python`

ANÁLISIS Y GUÍAS DE TERCEROS

Kyle Redelinghuys. *Claude Agent SDK: Subagents, Sessions and Why It's Worth It.* ksred.com

Towards AI. *Junior to Agent Architect: Mastering Anthropic's Claude SDK From Scratch.*

Let's Data Science. *Build Production AI Agents with Claude Agent SDK.*

Shivansh May. *Claude Agent SDK Deep Dive: What It Means to Use Claude Code as a Library.* Medium

Augment Code. *Claude Code vs Claude Agent SDK: Which Is for What.*

Augment Code. *Claude Agent SDK in Python: First Agent to Workflows.*

Abhinav Dobhal. *Building Autonomous AI Agents with the Claude Agent SDK.* Medium

Cobus Greyling. *Building Specialised AI Agents using Claude Agent SDK.* Medium

Lit Phansiri. *AI Agent Harness Comparison: Deep Agents or Claude Agent SDK.* Medium

LangChain. *Comparison with Claude Agent SDK.* `docs.langchain.com`

Dive into Claude Code: The Design Space of Today's and Future AI Agent Systems. arXiv

COBERTURA DE PRENSA Y COMUNIDAD

VentureBeat. *Anthropic reinstates OpenClaw and third-party agent usage on Claude subscriptions.*

PYMNTS. *Anthropic's Claude Subscription Shift Signals New AI Pricing Era.*

r/ClaudeAI. *A new monthly Agent SDK credit for Claude plans.*

r/ClaudeAI. */effort in one Claude Code instance silently nukes the prompt cache.*

Q QUIZ DE COMPRENSIÓN

¿Cuánto recuerdas?.

Diez preguntas para validar los conceptos clave. Respuesta inmediata con explicación. Tu progreso se guarda localmente.

PREGUNTA 01 DE 10

¿Cuál es la diferencia arquitectónica fundamental entre el SDK Estándar y el Agent SDK?

- El Agent SDK usa modelos más grandes
- El bucle de ejecución se invierte: en SDK Estándar lo controla Python; en Agent SDK reside dentro del SDK
- El Agent SDK soporta más lenguajes
- El SDK Estándar no permite herramientas

Incorrecto. En el paradigma 'LLM como Agente que Opera', el bucle vive dentro del SDK: planifica, llama herramientas, evalúa, y solo retorna al hilo de Python cuando termina, agota presupuesto o requiere intervención humana.

PREGUNTA 02 DE 10

¿Cuántos valores oficiales tiene permission_mode en el SDK Python?

- Tres: acceptEdits, dontAsk, auto
- Cuatro: default, acceptEdits, plan, bypassPermissions
- Cinco: default, acceptEdits, plan, dontAsk, bypassPermissions
- Seis incluyendo auto y deny

Incorrecto. El SDK Python expone cinco modos exactos: default, acceptEdits, plan, dontAsk, bypassPermissions. El valor 'auto' solo existe en TypeScript SDK; el PDF de respaldo lo lista erróneamente.

PREGUNTA 03 DE 10

Configuras allowed_tools=['Read'] y permission_mode='bypassPermissions'. ¿Qué bloquea Bash?

- Nada — bypassPermissions aprueba todo lo que llega al paso de modo
- Lo bloquea porque no está en allowed_tools
- Lo bloquea porque allowed_tools restringe estrictamente
- Solo Write quedaría permitido, no Bash

Incorrecto. allowed_tools no restringe — solo pre-aprueba. Como bypassPermissions aprueba todo en el paso 3 del orden de evaluación, las herramientas no listadas se aprueban igual. Para acotar de verdad: usa permission_mode='dontAsk' o disallowed_tools.

PREGUNTA 04 DE 10

¿Cuántos eventos de hooks soporta el SDK Python?

- Dos: PreToolUse y PostToolUse
- Cinco eventos principales
- Diez eventos
- Dieciocho (igual que TypeScript)

Incorrecto. El SDK Python soporta diez eventos: PreToolUse, PostToolUse, PostToolUseFailure, UserPromptSubmit, Stop, SubagentStop, SubagentStart, PreCompact, Notification, PermissionRequest. TypeScript SDK tiene 8 adicionales no disponibles aún en Python.

PREGUNTA 05 DE 10

¿Qué hace el SDK por defecto si NO configuras system_prompt?

- Carga el preset completo de Claude Code
- Usa un prompt mínimo solo para tool calling, sin guías Claude Code
- Lee CLAUDE.md del directorio automáticamente
- Lanza un error de configuración

Incorrecto. El default es un prompt mínimo que cubre solo tool calling. Esto difiere de 'claude -p' que sí usa el preset completo. Para activar el preset: system_prompt={'type': 'preset', 'preset': 'claude_code'}.

PREGUNTA 06 DE 10

Si quieres que el SDK cargue CLAUDE.md de tu proyecto, ¿qué necesitas?

- Nada, se carga automáticamente
- Configurar setting_sources=['project']
- Renombrarlo a .claude.md
- Usar el preset claude_code

Incorrecto. Por defecto setting_sources=None y el SDK NO carga ningún settings de filesystem. Para cargar CLAUDE.md hay que incluir 'project' explícitamente. Precedencia: local > project > user.

PREGUNTA 07 DE 10

Según la doc oficial de Anthropic, ¿qué soporta query() comparado con ClaudeSDKClient?

- query() no soporta hooks ni custom tools
- Ambos soportan hooks y custom tools; la diferencia es sesión, interrupciones y lifecycle
- Solo ClaudeSDKClient soporta MCP
- query() solo funciona con tools nativas

Incorrecto. La doc oficial (platform.claude.com/docs/en/agent-sdk/python) confirma que ambas interfaces soportan ✓ Hooks y ✓ Custom Tools. La diferencia real: query() crea sesión nueva cada vez; ClaudeSDKClient mantiene la misma sesión, soporta interrupciones (interrupt()), y permite cambiar permission_mode/modelo mid-sesión.

PREGUNTA 08 DE 10

Para reanudar una sesión específica preservando su historial, ¿qué usas?

- continue_conversation=True
- resume=session_id
- fork_session=True solo
- Nada — el SDK reanuda automáticamente

Incorrecto. resume=session_id reanuda una sesión específica por su ID. continue_conversation=True solo reanuda la MÁS RECIENTE del directorio. fork_session combinado con resume crea una sesión NUEVA con copia del historial.

PREGUNTA 09 DE 10

¿Para qué planes aplica el crédito mensual del Agent SDK (Junio 2026)?

- Solo Pro y Max
- Pro, Max, Team y Enterprise — pero NO Developer Platform con API key
- Todos los planes incluyendo API tradicional
- Solo Enterprise

Incorrecto. El crédito aplica a Pro ($20), Max 5× ($100), Max 20× ($200), Team Standard ($20), Team Premium ($100), Enterprise usage-based ($20), Enterprise seat-based Premium ($200). Es per-user, no pooleable, no rollover. NO aplica a Developer Platform con API key tradicional (sigue pay-as-you-go).

PREGUNTA 10 DE 10

¿Cuál es el error más caro al desplegar un agente autónomo en cron?

- Usar el modelo Haiku
- No definir max_turns o max_budget_usd
- Usar Bash en lugar de Python
- Habilitar WebSearch

Incorrecto. Sin guardrails, un fallo de parseo puede inducir al agente a intentar solventar el error eternamente, consumiendo la cuota mensual de tokens en horas durante la madrugada. max_turns y max_budget_usd son vitales para procesos desatendidos.

Francisco José Barros Cruz

EDICIÓN PÚBLICA · ELABORADO CON ASISTENCIA DE IA

https://www.linkedin.com/in/francisco-jose-barros-cruz/ (https://www.linkedin.com/in/francisco-jose-barros-cruz/)