EcomLabs

GUÍA PASO A PASO

Function calling explicado con un carrito

El ejemplo completo en Node y TypeScript: tres herramientas de carrito con strict: true, el ciclo tool_use y tool_result, errores con is_error, una prueba que corre sin API key y la checklist de seguridad.

PARA	Tiendas y equipos que quieren que un asistente de IA busque productos, llene el carrito y cotice envíos
PROBADO CON	Node 22.22.3 · @anthropic-ai/sdk 1.6.0 y tsc --strict · cliente simulado con respuestas de la forma documentada, sin llamar a la API
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	platform.claude.com/docs/en/agents-and-tools/tool-use/overview
ARTÍCULO	ecomlabs.dev/g/tools
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a armar
Paso 01	El archivo completo
Paso 02	Probarlo y usarlo
Al terminar	Checklist de seguridad

ANTES DE EMPEZAR

Qué vas a armar

Un asistente de compras con function calling: el modelo busca productos, los agrega al carrito y calcula el envío llamando tres funciones de tu código. Todo cabe en `tienda.ts`. Lo probamos en Node 22.22.3, que ejecuta archivos `.ts` directamente, con `@anthropic-ai/sdk` 1.6.0. En una carpeta nueva: `npm init -y`, `npm pkg set type=module` y `npm install @anthropic-ai/sdk`. Las formas de la API siguen la documentación de tool use de Anthropic (19 de noviembre de 2026); catálogo, precios y regla de envío son de ejemplo.

PASO 01

El archivo completo

Guarda esto como `tienda.ts`: el catálogo de ejemplo, las tres herramientas con `strict: true`, la función que las ejecuta y valida, y el ciclo.

```
// tienda.ts: function calling with three cart tools (Node 22 + @anthropic-ai/sdk)
import Anthropic from "@anthropic-ai/sdk";

// Sample catalog, prices in MXN. In your store it comes from the database.
const CATALOGO = [
  { sku: "TAZA-350", nombre: "Taza de cerámica 350 ml", precio: 189, stock: 12 },
  { sku: "CAFE-500", nombre: "Café de Veracruz 500 g", precio: 245, stock: 0 },
  { sku: "PRENSA-1L", nombre: "Prensa francesa 1 L", precio: 649, stock: 3 },
];
const MAX_POR_PRODUCTO = 10;
const MAX_VUELTAS = 6;
export type Carrito = Map<string, number>; // sku -> quantity, one per session

// Strict schema: every field required, no extra fields.
const herramienta = (name: string, description: string, properties: Record<string, object>): Anthropic.Tool =>
({
  name,
  description,
  strict: true,
  input_schema: { type: "object", properties, required: Object.keys(properties), additionalProperties: false
},
});

export const tools: Anthropic.Tool[] = [
  herramienta(
    "buscar_producto",
    "Busca productos del catálogo por nombre. Devuelve hasta 5 resultados con sku, nombre, precio en MXN y
unidades disponibles. Úsala antes de agregar algo al carrito, para obtener el sku exacto. No consulta pedidos
ni el carrito.",
    { consulta: { type: "string", description: "Texto a buscar, por ejemplo: taza" } },
  ),
  herramienta(
```

```
    "agregar_al_carrito",
    "Agrega un producto al carrito de la sesión actual. Necesita el sku exacto que devolvió buscar_producto y una cantidad entera de 1 a 10. Si no hay stock suficiente, devuelve un error con las unidades disponibles. El precio lo pone la tienda, no se envía.",
    {
      sku: { type: "string", description: "SKU exacto, por ejemplo TAZA-350" },
      cantidad: { type: "integer", description: "Unidades a agregar, de 1 a 10" },
    },
  ),
  herramienta(
    "calcular_envio",
    "Calcula el envío del carrito actual a un código postal de México. Úsala solo cuando el carrito tenga productos. Devuelve subtotal, envío y total en MXN. No crea el pedido ni cobra nada.",
    { codigo_postal: { type: "string", description: "Código postal de 5 dígitos" } },
  ),
];

// An error the model can fix: it goes back as a tool_result with is_error.
class ToolError extends Error {}

const subtotal = (carrito: Carrito) =>
  [...carrito].reduce((s, [sku, n]) => s + CATALOGO.find((p) => p.sku === sku)!.precio * n, 0);

function ejecutar(nombre: string, input: unknown, carrito: Carrito): string {
  if (typeof input !== "object" || input === null) throw new ToolError("La entrada debe ser un objeto.");
  const args = input as Record<string, unknown>;
  switch (nombre) {
    case "buscar_producto": {
      const q = String(args.consulta ?? "").trim().toLowerCase();
      if (!q) throw new ToolError("La consulta está vacía.");
      const hallados = CATALOGO.filter((p) => p.nombre.toLowerCase().includes(q)).slice(0, 5);
      return JSON.stringify(hallados.map(({ sku, nombre, precio, stock }) => ({ sku, nombre, precio, disponibles: stock })));
    }
    case "agregar_al_carrito": {
      const producto = CATALOGO.find((p) => p.sku === args.sku);
      if (!producto) throw new ToolError(`El sku ${String(args.sku)} no existe. Usa buscar_producto para obtenerlo.`);
      const cantidad = args.cantidad;
      if (typeof cantidad !== "number" || !Number.isInteger(cantidad) || cantidad < 1 || cantidad > MAX_POR_PRODUCTO) {
        throw new ToolError(`La cantidad debe ser un entero de 1 a ${MAX_POR_PRODUCTO}.`);
      }
      const total = (carrito.get(producto.sku) ?? 0) + cantidad;
      if (total > producto.stock) throw new ToolError(`Solo hay ${producto.stock} unidades de ${producto.sku}.`);
      carrito.set(producto.sku, total);
      return JSON.stringify({ sku: producto.sku, cantidad: total, subtotal: subtotal(carrito) });
    }
    case "calcular_envio": {
      const cp = String(args.codigo_postal ?? "");
```

```
      if (!/^\d{5}$/.test(cp)) throw new ToolError("El código postal debe tener 5 dígitos.");
      if (carrito.size === 0) throw new ToolError("El carrito está vacío.");
      const sub = subtotal(carrito);
      const envio = sub >= 999 ? 0 : 99; // illustrative rule: free shipping from 999 MXN
      return JSON.stringify({ codigo_postal: cp, subtotal: sub, envio, total: sub + envio });
    }
    default:
      throw new ToolError(`No existe la herramienta ${nombre}.`);
  }
}

const SYSTEM =
  "Eres el asistente de compras de una tienda en línea. Usa las herramientas para buscar productos, agregarlos
al carrito y calcular el envío. No inventes precios ni sku. Si una herramienta devuelve un error, corrige la
llamada o explícale al cliente qué pasó.";

export async function conversar(client: Anthropic, pregunta: string, carrito: Carrito) {
  const messages: Anthropic.MessageParam[] = [{ role: "user", content: pregunta }];

  for (let vuelta = 0; vuelta < MAX_VUELTAS; vuelta++) {
    const res = await client.messages.create({
      model: "claude-opus-5-5",
      max_tokens: 16000,
      system: SYSTEM,
      tools,
      messages,
    });

    if (res.stop_reason !== "tool_use") {
      // end_turn is the normal answer; refusal or max_tokens go back to the caller.
      const texto = res.content.flatMap((b) => (b.type === "text" ? [b.text] : [])).join("\n");
      return { texto, stop: res.stop_reason, messages };
    }

    messages.push({ role: "assistant", content: res.content }); // unchanged, thinking blocks included
    const resultados: Anthropic.ToolResultBlockParam[] = [];
    for (const bloque of res.content) {
      if (bloque.type !== "tool_use") continue;
      try {
        const salida = ejecutar(bloque.name, bloque.input, carrito);
        resultados.push({ type: "tool_result", tool_use_id: bloque.id, content: salida });
      } catch (err) {
        if (!(err instanceof ToolError)) console.error(bloque.name, err); // log the real cause, don't send it
        const mensaje = err instanceof ToolError ? err.message : "Error interno de la tienda. Intenta más
tarde.";
        resultados.push({ type: "tool_result", tool_use_id: bloque.id, content: mensaje, is_error: true });
      }
    }
    messages.push({ role: "user", content: resultados }); // every result, in one message
  }
```

```
  throw new Error(`Sin respuesta final después de ${MAX_VUELTAS} vueltas.`);
}
```

- `ToolError` separa lo que el modelo puede corregir (vuelve con su mensaje) de tus fallos (vuelven con un mensaje genérico y el detalle va a tus logs).
- El carrito es un parámetro de `conversar`, no de las herramientas: lo pone tu servidor a partir de la sesión.
- `MAX_POR_PRODUCTO` vive en el código porque el modo estricto no admite `minimum` ni `maximum` (la API responde 400).

PASO 02

Probarlo y usarlo

Sin gastar. El cliente es un parámetro, así que en una prueba con `node --test` le pasas uno falso: `{ messages: { create: async () => respuestas.shift()! } } as unknown as Anthropic`, donde `respuestas` son mensajes con la forma documentada (`stop_reason: "tool_use"` y bloques `tool_use` con `id`, `name` e `input`). Prueba el caso feliz (dos tazas, total 378 + 99) y los errores: cantidad 50, un sku inventado, un código postal de cuatro dígitos y un modelo que nunca deja de pedir herramientas.

Con la API. `await conversar(new Anthropic(), "Agrega dos tazas y dime el envío al 06700", carrito)`, con la clave en `ANTHROPIC_API_KEY` y un carrito por sesión. Revisa `stop` antes de mostrar `texto`: con `refusal` o `max_tokens`, decide tú qué ve el cliente. No fuerces la herramienta con `tool_choice` en `any` o `tool`: en Claude Opus 5.5 devuelve 400.

> **CUIDADO**
>
> Una prueba con cliente falso comprueba tu código, no al modelo. Antes de abrirlo a clientes, corre conversaciones reales en un entorno de prueba y revisa en tus logs qué herramientas pidió y con qué argumentos.

AL TERMINAR

Checklist de seguridad

- [ ] Precio, descuentos, carrito y cliente salen de tu servidor; ninguna herramienta los recibe del modelo.
- [ ] Cada `input` se trata como entrada de usuario: el sku existe, la cantidad está en rango, hay stock (y la sobreventa se resuelve al pagar).
- [ ] Ninguna herramienta cobra, reembolsa ni confirma pedidos; el cliente paga en el checkout.
- [ ] Los errores vuelven con `is_error: true`; los internos, sin stack traces ni claves.
- [ ] Lo que escriben terceros (reseñas, proveedores) va solo en `tool_result`, como no confiable.
- [ ] El ciclo tiene tope de vueltas y solo ejecuta herramientas cuando `stop_reason` es `tool_use`.
- [ ] Hay un límite de peticiones por sesión, registras cada llamada y la clave de la API vive solo en el servidor.