GUÍA PASO A PASO

Prompt caching en la API de Claude: el ejemplo completo

La petición con el bloque fijo marcado con cache_control, una función que calcula el costo de cada llamada con los campos de uso, pruebas con un cliente simulado que no gastan dinero y la checklist de orden del prompt.

PARA	Apps en Node o TypeScript que mandan a la API de Claude un prompt largo y repetido
PROBADO CON	Node 22.22.3 · @anthropic-ai/sdk 1.10.0 y tsc --strict · cliente simulado, sin llamadas a la API
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	platform.claude.com/docs/en/build-with-claude/prompt-caching
ARTÍCULO	ecomlabs.dev/g/cache
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué necesitas
Paso 01	La petición con el bloque marcado
Paso 02	El costo de cada llamada
Paso 03	Probarlo sin llamar a la API
Paso 04	Leer el resultado
Al terminar	Checklist de orden del prompt

ANTES DE EMPEZAR

Qué necesitas

- Node 22 o posterior, `npm install @anthropic-ai/sdk` y `"type": "module"` en `package.json`. Node 22.22.3 ejecutó estos `.ts` sin compilar.
- Para `tsc`: `"strict": true`, `"module": "NodeNext"` y `"allowImportingTsExtensions": true`, que permite importar `./ask.ts` con su extensión.
- La clave en la variable `ANTHROPIC_API_KEY`, nunca en el código.
- Un `policies.md` con la parte fija, por encima del mínimo del modelo: 512 tokens en Claude Opus 5.5.

Precios de Opus 5.5 por millón de tokens el 19 de noviembre de 2026: entrada $4, escritura de 5 minutos $5, de 1 hora $8, lectura $0.20, salida $20.

PASO 01

La petición con el bloque marcado

Lo fijo va en `system`, leído una vez al arrancar, con `cache_control` en su último bloque. La pregunta va en `messages`, después del corte:

```
// ask.ts
import { readFileSync } from "node:fs";
import type Anthropic from "@anthropic-ai/sdk";

export const MODEL = "claude-opus-5-5";

// Fixed part: read once at startup, the same bytes on every call.
const POLICIES = readFileSync(new URL("./policies.md", import.meta.url), "utf8");

export const SYSTEM: Anthropic.TextBlockParam[] = [
  { type: "text", text: "You answer customer questions for an online store. Use only the policies below." },
  // Breakpoint: everything up to and including this block is cached.
  { type: "text", text: POLICIES, cache_control: { type: "ephemeral" } },
];

// Variable part: the question goes last, after the breakpoint.
export function ask(client: Anthropic, question: string) {
  return client.messages.create({
    model: MODEL,
    max_tokens: 1024,
    system: SYSTEM,
    messages: [{ role: "user", content: question }],
  });
}
```

Para 1 hora: `cache_control: { type: "ephemeral", ttl: "1h" }`. Con tools largas, pon otro punto en la última tool. Máximo 4 puntos por petición.

PASO 02

El costo de cada llamada

`cost()` aplica la fórmula de la documentación: total de entrada = lectura + escritura + `input_tokens`. `run.ts` hace dos preguntas e imprime una línea por llamada; con una clave real, la primera debe escribir y la segunda leer.

```
// cost.ts
import type Anthropic from "@anthropic-ai/sdk";

// USD per million tokens for Claude Opus 5.5, from the pricing page on 2026-11-19.
export const PRICE = { input: 4, write5m: 5, write1h: 8, read: 0.2, output: 20 };

export function cost(u: Anthropic.Usage, p = PRICE) {
  const written = u.cache_creation_input_tokens ?? 0;
  const read = u.cache_read_input_tokens ?? 0;
  const w1h = u.cache_creation?.ephemeral_1h_input_tokens ?? 0;
  const usd =
    (u.input_tokens * p.input + (written - w1h) * p.write5m + w1h * p.write1h +
      read * p.read + u.output_tokens * p.output) / 1e6;
  const total = read + written + u.input_tokens; // the docs' total input formula
  return { usd, total, hitRate: total ? read / total : 0, cached: written + read > 0 };
}
```

```
// run.ts
import Anthropic from "@anthropic-ai/sdk";
import { ask } from "./ask.ts";
import { cost } from "./cost.ts";

const client = new Anthropic(); // reads ANTHROPIC_API_KEY from the environment

for (const q of ["Do you ship to Monterrey?", "Can I return a gift?"]) {
  const { usage } = await ask(client, q);
  const c = cost(usage);
  console.log(
    `write ${usage.cache_creation_input_tokens} · read ${usage.cache_read_input_tokens}` +
      ` · new ${usage.input_tokens} · $${c.usd.toFixed(4)}`,
  );
  if (!c.cached) console.warn("no cache: prefix under the model minimum, or no breakpoint");
}
```

PASO 03

Probarlo sin llamar a la API

El cliente simulado aplica la regla documentada (prefijo idéntico y por encima del mínimo) y cuenta tokens como caracteres entre cuatro: prueba la forma de tu petición, no los números de Claude. Se ejecuta con `node --test ask.test.ts`, sin clave; las cuatro pruebas pasaron en Node 22.22.3.

```ts
// ask.test.ts — node --test ask.test.ts (no API call, no key)
import { test } from "node:test";
import assert from "node:assert/strict";
import type Anthropic from "@anthropic-ai/sdk";
import { ask, SYSTEM } from "./ask.ts";
import { cost } from "./cost.ts";

// Fake client with the documented rule: the prefix up to the breakpoint must be
// identical and over the minimum (512 tokens on Opus 5.5). Tokens = chars / 4.
function fake(min = 512) {
  const seen = new Set<string>();
  const create = async (body: Anthropic.MessageCreateParamsNonStreaming) => {
    const sys = body.system as Anthropic.TextBlockParam[];
    const prefix = sys.slice(0, sys.findIndex((b) => b.cache_control) + 1);
    const size = Math.ceil(prefix.map((b) => b.text).join("").length / 4);
    const ok = size > 0 && size >= min;
    const key = JSON.stringify(prefix);
    const read = ok && seen.has(key) ? size : 0;
    const write = ok && !read ? size : 0;
    if (ok) seen.add(key);
    return { usage: { input_tokens: ok ? 20 : size + 20, output_tokens: 100,
      cache_creation_input_tokens: write, cache_read_input_tokens: read, cache_creation: null } };
  };
  return { messages: { create } } as unknown as Anthropic;
}

test("first call writes the prefix, second call reads it", async () => {
  const client = fake();
  const a = (await ask(client, "Do you ship to Monterrey?")).usage;
  const b = (await ask(client, "Can I return a gift?")).usage;
  assert.ok(a.cache_creation_input_tokens! > 512 && a.cache_read_input_tokens === 0);
  assert.equal(b.cache_read_input_tokens, a.cache_creation_input_tokens);
  assert.ok(cost(b).usd < cost(a).usd);
});

test("a timestamp before the breakpoint means no read, ever", async () => {
  const client = fake();
  const stamped = (t: string) => client.messages.create({
    model: "claude-opus-5-5", max_tokens: 1024,
    system: [{ type: "text", text: `Now: ${t}` }, ...SYSTEM],
    messages: [{ role: "user", content: "hi" }],
  });
  await stamped("10:00:01");
  assert.equal((await stamped("10:00:02")).usage.cache_read_input_tokens, 0);
});

test("a prefix under the minimum is not cached", async () => {
  assert.equal(cost((await ask(fake(1e9), "hi")).usage).cached, false);
});
```

```
test("cost() matches the price table by hand", () => {
  const u = { input_tokens: 200, output_tokens: 300, cache_creation_input_tokens: 0,
    cache_read_input_tokens: 20_000, cache_creation: null } as Anthropic.Usage;
  // 200 × 4 + 20,000 × 0.2 + 300 × 20 = 10,800 per million tokens
  assert.equal(cost(u).usd.toFixed(4), "0.0108");
});
```

PASO 04

Leer el resultado

- **Escritura y luego lecturas:** funciona.
- **Escritura en cada llamada:** algo cambia antes del punto (fecha, id, orden de las tools) o pasaron más de 5 minutos.
- **Escritura y lectura en 0:** el prefijo no llega al mínimo o no hay `cache_control`. No hay error.

> **CUIDADO**
>
> Cambiar `tool_choice`, las imágenes o los parámetros de razonamiento invalida al menos los mensajes cacheados; cambiar las tools invalida todo.

AL TERMINAR

Checklist de orden del prompt

- [ ] 1. Tools, siempre en el mismo orden y con el mismo JSON.
- [ ] 2. Instrucciones, sin fecha, hora, id de sesión ni nombre de usuario.
- [ ] 3. Documentos largos: políticas, catálogo, base de conocimiento pequeña.
- [ ] 4. `cache_control` en el último bloque fijo.
- [ ] 5. Historial (con otro punto si crece mucho, o caché automática).
- [ ] 6. Fragmentos de RAG de esta pregunta, fecha y datos del usuario.
- [ ] 7. La pregunta nueva, al final.
- [ ] La segunda llamada muestra `cache_read_input_tokens` mayor que 0.