EcomLabs

STEP-BY-STEP GUIDE

Prompt caching in the Claude API: the complete example

The request with its fixed block marked with cache_control, a function that prices every call from the usage fields, tests against a fake client that cost nothing to run, and the prompt order checklist.

FOR	Node or TypeScript apps that send a long, repeated prompt to the Claude API
TESTED WITH	Node 22.22.3 · @anthropic-ai/sdk 1.10.0 and tsc --strict · fake client, no API calls
UPDATED	November 19, 2026
DOCS	platform.claude.com/docs/en/build-with-claude/prompt-caching
ARTICLE	ecomlabs.dev/g/cache-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you need
Step 01	The request with the marked block
Step 02	The cost of each call
Step 03	Test it without calling the API
Step 04	Read the result
When you finish	Prompt order checklist

BEFORE YOU START

What you need

- Node 22 or later, `npm install @anthropic-ai/sdk` and `"type": "module"` in `package.json`. Node 22.22.3 ran these `.ts` files with no build step.
- For `tsc`: `"strict": true`, `"module": "NodeNext"` and `"allowImportingTsExtensions": true`, which lets you import `./ask.ts` with its extension.
- Your key in the `ANTHROPIC_API_KEY` environment variable, never in the code.
- A `policies.md` with the fixed part, over the model's minimum: 512 tokens on Claude Opus 5.5.

Opus 5.5 prices per million tokens on November 19, 2026: input $4, 5-minute write $5, 1-hour write $8, cache read $0.20, output $20.

STEP 01

The request with the marked block

The fixed part goes in `system`, read once at startup, with `cache_control` on its last block. The question goes in `messages`, after the breakpoint:

```
// 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 }],
  });
}
```

For 1 hour: `cache_control: { type: "ephemeral", ttl: "1h" }`. With long tool definitions, add a breakpoint on the last tool. At most 4 breakpoints per request.

STEP 02

The cost of each call

`cost()` applies the docs' formula: total input = read + creation + `input_tokens`. `run.ts` asks two questions and prints one line per call; with a real key, the first should write and the second read.

```
// 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");
}
```

STEP 03

Test it without calling the API

The fake client applies the documented rule (identical prefix, over the minimum) and counts tokens as characters divided by four: it tests the shape of your request, not Claude's numbers. Run it with `node --test ask.test.ts`, no key needed; all four tests passed on 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");
});
```

STEP 04

Read the result

- **A write, then reads:** it works.
- **A write on every call:** something before the breakpoint changes (a date, an id, the tool order), or more than 5 minutes passed.
- **Write and read both 0:** the prefix is under the minimum or there is no `cache_control`. No error is returned.

> **CAUTION**
> Changing `tool_choice`, images or thinking parameters invalidates at least the cached messages; changing the tools invalidates everything.

WHEN YOU FINISH

Prompt order checklist

- [ ] 1. Tools, always in the same order with the same JSON.
- [ ] 2. Instructions, with no date, time, session id or user name.
- [ ] 3. Long documents: policies, catalog, a small knowledge base.
- [ ] 4. `cache_control` on the last fixed block.
- [ ] 5. History (with another breakpoint if it grows long, or automatic caching).
- [ ] 6. This question's RAG chunks, the date and user data.
- [ ] 7. The new question, last.
- [ ] The second call shows `cache_read_input_tokens` above 0.