FREE PREVIEW · ~32% OF THE FULL GUIDE

Unofficial Study Guide for the CCDV-F Exam

Claude Certified Developer — Foundations · sample edition

CCDV-F | 53 Q · 120 MIN | PASS 720/1000

An independent study resource by Zehon.com · August 2026
This free preview contains complete, unabridged sections of the full guide: Day 1 of the 7-day plan, all of Domain 1, and 5 practice questions with full answer explanations.

The 7-day plan

Read a domain, get your hands dirty, checkpoint with the quiz — every day.

~2 hours a day for 7 days. Every day pairs reading with hands-on API work — CCDV-F rewards muscle memory for parameter names, event orders, and pricing math. Keep the study guide open every day and end each day with its checkpoint; if you miss a checkpoint item, reread that guide section before moving on.

Prereqs: Python 3.10+ (or Node 18+), an Anthropic API key with a few dollars of credit (`export ANTHROPIC_API_KEY=...`), `pip install anthropic`, and Claude Code installed (`npm install -g @anthropic-ai/claude-code`).

Materials: the study guide (open every day) · the 36-question quiz bank · a scratch repo · an API key with a few dollars of credit

Domain-weight order: Applications & Integration (33%) and Model Selection (17%) are half the exam — they get days 1–3. The quiz on day 7 tells you whether to book.

Day 1 — API mechanics & model family (Domains 1 + 2 core)

Read (45 min): Guide Domain 1 (API Mechanics, App Design) + Domain 2 model table. Skim docs: Messages API reference, models overview.

Hands-on (60 min):

- Make a minimal `client.messages.create()` call on `claude-haiku-4-5`. Print `id`, `stop_reason`, `usage`.
- Force each stop reason: set `max_tokens=10` (→ `max_tokens`), add `stop_sequences=["5"]` and ask it to count (→ `stop_sequence`), normal call (→ `end_turn`).
- Send the same prompt to Haiku, Sonnet 5, and Opus 5; record latency and `usage`, then compute the dollar cost of each from the pricing table by hand.
- Break things on purpose: wrong API key (observe the 401 JSON error shape), missing `max_tokens` (400). Note `request_id`.

✓ **CHECKPOINT (15 MIN)**

From memory: the 3 required request fields; all 7 stop_reason values; 429 vs 529 vs 500 and which you retry; prices for all three models.

🔒 **IN THE FULL EDITION — THE REST OF THE PLAN**

- Day 2 — Streaming, errors/retries, rate limits (Domain 1)
- Day 3 — Batches & prompt caching (Domains 1 + 2 cost)
- Day 4 — Tool use, structured output, thinking/effort (Domains 5 + 4 + 2)
- Day 5 — Agents, Agent SDK, MCP (Domains 3 + 5)
- Day 6 — Claude Code, hooks, security (Domains 7 + 6 + 8)
- Day 7 — Quick reference, full quiz, logistics

The study guide

Eight weighted domains, twenty-five objectives — implementation facts you either know or you don't.

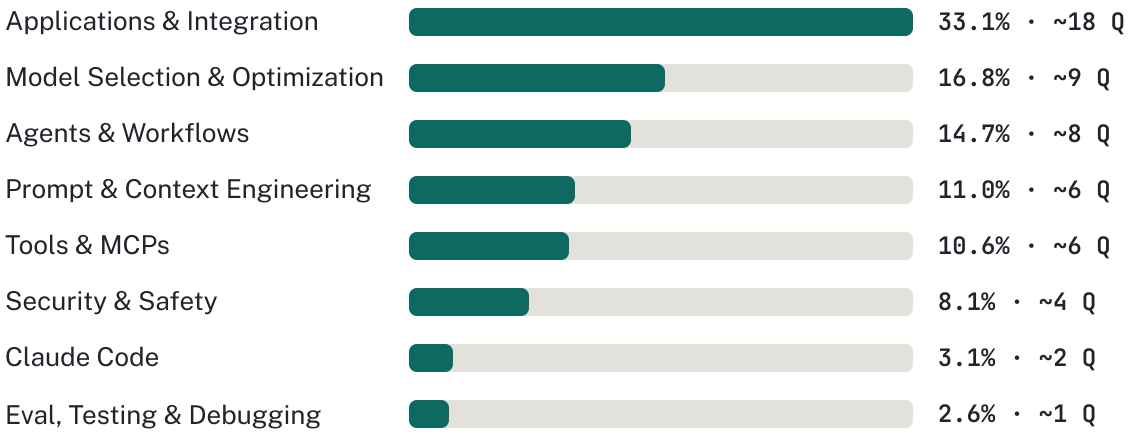
Everything on this page maps to the 25 objectives of the official exam blueprint, organized into its 8 published weighted domains. The material is implementation-focused — "you either know it or you don't": exact parameter names, model trade-offs, and API-surface decisions. Facts verified against docs.claude.com / platform.claude.com / code.claude.com (August 2026). With 120 minutes for 53 questions, a prepared developer has ample time.

EXAM SNAPSHOT

53 Q IMPLEMENTATION-FOCUSED	**8 domains** 25 OBJECTIVES	**120 min** CLOSED BOOK	**720** PASS, 100–1000 SCALE
$125 PEARSON ONVUE	**0** GUESSING PENALTY — ANSWER ALL		

Online proctored via Pearson OnVUE. Credly badge valid 12 months. Retakes: 14 days after the 1st attempt, 30 after the 2nd, 90 after the 3rd; max 4 attempts per rolling 12 months. Logistics reflect the official program pages as of August 2026 — verify before booking.

DOMAIN WEIGHT · APPROXIMATE QUESTIONS OF 53

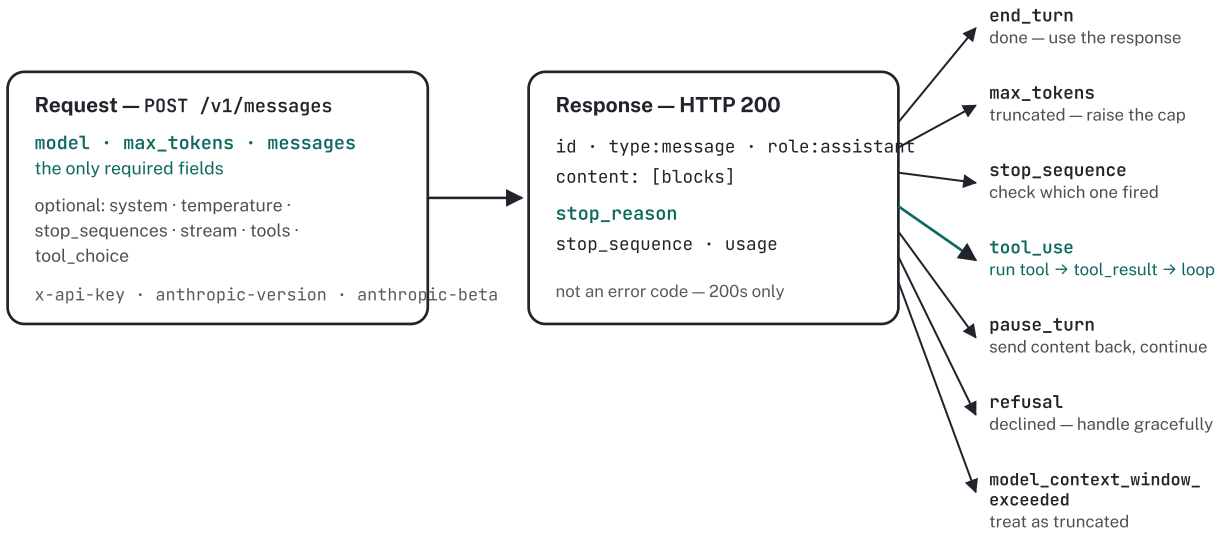


Where the 53 questions come from. Domain 1 alone is a third of the exam; Domains 1+2 together are half — study in this order.

Domain 1 · Applications and Integration 33.1% · ~18 QUESTIONS

The heaviest domain by far. Owns the Messages API surface: request/response anatomy, stop reasons, errors, rate limits, streaming, batches, and app-design decisions.

1.1 · Claude API mechanics: request and response



The full request→response cycle and all 7 `stop_reason` exits. `stop_reason` appears on *successful* HTTP 200 responses — it is not an error code.

Request anatomy (Messages API, `POST /v1/messages`):

```
{
  "model": "claude-sonnet-5",
  "max_tokens": 1024,
  "system": "You are a concise assistant.",
  "messages": [
    {"role": "user", "content": "Hello"}
  ],
  "temperature": 1.0,
  "stop_sequences": ["###"],
  "stream": false,
  "tools": [],
  "tool_choice": {"type": "auto"}
}
```

★ **REQUIRED VS OPTIONAL — MEMORIZE THESE**

Required fields: `model`, `max_tokens`, `messages`. Everything else is optional. Auth headers: `x-api-key: sk-ant-...` and `anthropic-version: 2023-06-01`; beta features add `anthropic-beta: <header>`.

- Only two conversational roles in `messages`: `user` and `assistant`. There is **no** `tool` **role** — tool results go back in a `user` message as `tool_result` content blocks.
- The system prompt is normally the top-level `system` field, not a message.
- **The API is stateless.** You must resend the *full conversation history* every request; the model keeps no state between calls.
- Messages alternate user/assistant; the last message before a response is typically `user`. An `assistant` message placed last acts as a **prefill** (Claude continues from it).

Response anatomy:

```
{
  "id": "msg_01...",
  "type": "message",
  "role": "assistant",
  "model": "claude-sonnet-5",
  "content": [{"type": "text", "text": "Hi!"}],
  "stop_reason": "end_turn",
  "stop_sequence": null,
  "usage": {"input_tokens": 12, "output_tokens": 6,
            "cache_creation_input_tokens": 0, "cache_read_input_tokens": 0}
}
```

The 7 `stop_reason` values (memorize all):

STOP_REASON	MEANING	YOUR ACTION
`end_turn`	Finished naturally	Use the response
`max_tokens`	Hit your `max_tokens` cap	Response truncated — raise the cap or continue
`stop_sequence`	Emitted one of your `stop_sequences`	Check the `stop_sequence` field for which one
`tool_use`	Wants a tool executed	Run the tool, return a `tool_result`, loop

`pause_turn`	Server-tool loop paused (long-running server tools)	Send the assistant content back to continue
`refusal`	Declined for safety reasons	Handle gracefully / adjust the request
`model_context_window_exceeded`	Output filled the context window	Treat as truncated

1.2 · HTTP errors, retries, and rate limits

HTTP errors (memorize the table):

HTTP	ERROR.TYPE	NOTES
400	`invalid_request_error`	Malformed request
401	`authentication_error`	Bad/revoked API key
402	`billing_error`	Billing/payment issue
403	`permission_error`	Key lacks permission for the resource
404	`not_found_error`	Resource not found
409	`conflict_error`	State conflict
413	`request_too_large`	> 32 MB Messages request (256 MB batch, 500 MB Files)
429	`rate_limit_error`	Rate limit or spend cap hit
500	`api_error`	Internal error — retry with backoff
504	`timeout_error`	Request timed out
529	`overloaded_error`	API temporarily overloaded — back off and retry

Error shape: `{"type": "error", "error": {"type": "...", "message": "..."}, "request_id": "req_..."}`. Every response carries a `request-id` header — quote it to support.

> ⚖ **RETRY POLICY**
>
> Retry **429/500/529** (and timeouts) with **exponential backoff + jitter**; honor the `retry-after` header on 429. Official SDKs retry transient failures automatically (default **2 retries**). Do NOT blindly retry 400/401/403 — fix the request or key instead.

Rate limits are enforced per model class on three dimensions using a **token-bucket** algorithm (continuous replenishment, allows bursts). Limits rise with usage tier.

`RPM — requests/min` `ITPM — input tokens/min` `OTPM — output tokens/min`

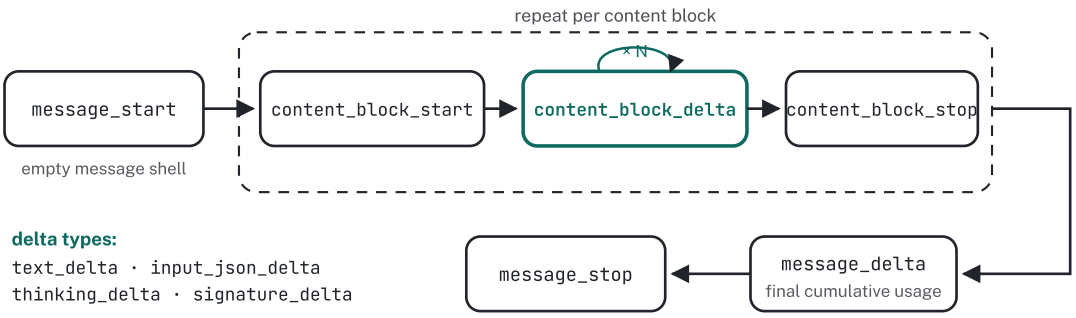
Response headers: `anthropic-ratelimit-requests-limit/-remaining/-reset`, plus the input/output token variants, and `retry-after` on 429. **Cached (read) input tokens do not count toward ITPM** on current models — caching raises effective throughput.

1.3 · Claude app design: the surface decision matrix

The core surface decision — expect several scenario questions:

NEED	USE
Interactive chat / user waiting	Messages API (real-time), usually **streaming**
Large offline volume, deadline ≥ hours	**Message Batches API** (50% cheaper)
Long single responses (large `max_tokens`)	**Streaming required** to avoid network timeouts
Agent that plans, uses tools, edits files	**Claude Agent SDK** (runs the loop for you)
You implement the tool loop yourself	Client SDK / raw Messages API
Terminal / CI usage	Claude Code CLI (`claude -p`)

1.4 · Streaming (SSE)



Also on the wire: `ping` keep-alives and in-stream `error` events (e.g. overloaded_error). Set `"stream": true`. `input_json_delta` delivers tool arguments as partial JSON strings — assemble before parsing.

The exact SSE event order: `message_start` → per block `content_block_start` → `content_block_delta` × N → `content_block_stop` → `message_delta` → `message_stop`. Final token usage arrives on the last `message_delta`.

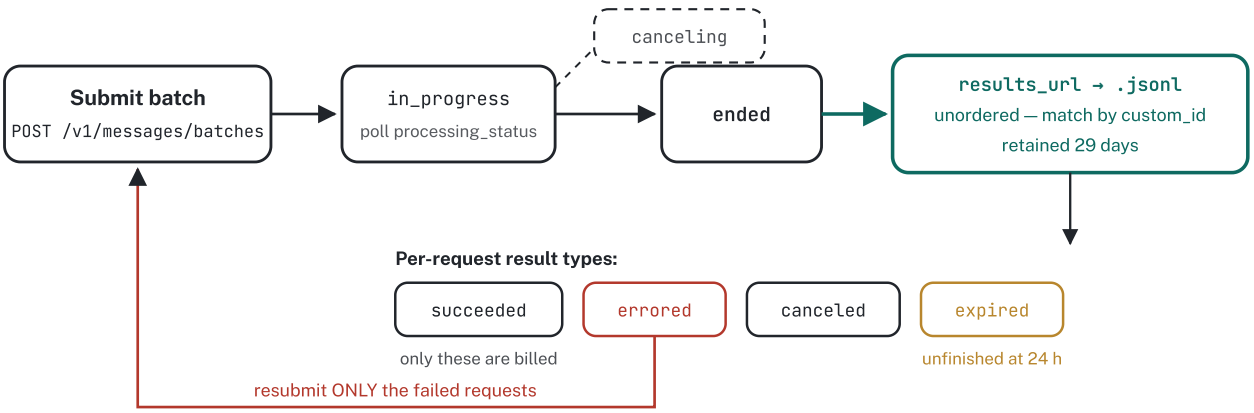
WHEN TO STREAM

- Real-time UX (user watching tokens)
- Long outputs
- Avoiding HTTP timeouts — recommended/required for long-running requests

WHEN NOT TO

- Batch jobs
- Simple short calls
- Pipelines that only need the final result

1.5 · Message Batches API



Most batches finish < 1 hour; the hard window is **24 hours**, then unfinished requests **expire**.
SLA math: to guarantee output by deadline D, submit no later than **D − 24 h**.

Batch lifecycle: submit → poll → `ended` → fetch `.jsonl` results and match by `custom_id`. Failure pattern: find the errored requests by `custom_id` and resubmit only those.

Batch mechanics (core material):

FACT	VALUE
Endpoint	`POST /v1/messages/batches`
Discount	**50% off** input AND output tokens
Batch size cap	**100,000 requests or 256 MB**
Processing	most finish < 1 hour; hard window **24 hours**, then unfinished requests **expire**
Results retention	**29 days**, fetched from `results_url` as **.jsonl**
Correlation	`custom_id` per request (results are NOT returned in order)

`processing_status`	`in_progress` → (`canceling`) → `ended`
Per-request result types	`succeeded`, `errored`, `canceled`, `expired` (only `succeeded` is billed)
Supported	tools, vision, system prompts, extended thinking, prompt caching (best-effort hits)
Not supported	`stream: true`, fast mode (`speed`)

1.6 · Software engineering foundations

- Use the **official SDKs** (Python `anthropic`, TypeScript `@anthropic-ai/sdk`); they handle auth headers, retries (exponential backoff, 2 by default, configurable via `max_retries`), and streaming parsing.
- Idempotency in your own pipeline: track work by stable IDs (mirrors `custom_id` in batches).
- Timeouts: long requests (>10 min) should use streaming or batches; keep TCP keep-alives on.
- Version pinning: use dated model IDs (e.g., `claude-haiku-4-5-20251001`) in production for reproducibility; bare aliases (`claude-sonnet-5`) track the latest snapshot.
- Log `request_id`, `stop_reason`, and `usage` on every call — the debugging trio.

1.7 · Config management

- **API keys from environment variables** (`ANTHROPIC_API_KEY`) or a secret manager — never hard-coded, never committed.
- Separate keys/workspaces per environment (dev/staging/prod); workspace-level spend and rate limits protect shared orgs.
- Model choice, `max_tokens`, temperature, and prompts are config — externalize them so you can change without redeploying.
- Claude Code settings precedence (see Domain 7): managed policy > local project (`.claude/settings.local.json`) > project (`.claude/settings.json`) > user (`~/.claude/settings.json`).
- `.mcp.json` supports `${VAR}` / `${VAR:-default}` expansion so secrets stay in the environment.

1.8 · Understanding requirements

Translate requirements into API-surface decisions:

- "User is waiting" → real-time + streaming. "By tomorrow morning" → batch. "Cheapest possible bulk job" → batch + Haiku + caching.
- Latency requirement → smaller model, lower effort, fast mode (Opus), streaming for perceived latency.
- Accuracy/reasoning requirement → larger model, higher effort/thinking.
- Guaranteed output shape → structured outputs / strict tool use, not "please answer in JSON".
- Compliance rules that must ALWAYS hold → code (hooks/validators), never prompt instructions alone.

1.9 · Systems life cycle

Lifecycle: prototype (Console/Workbench) → evaluate on a test set → ship → monitor (usage, cost, error rates, quality) → iterate prompts/models → re-evaluate before promoting.

- Model deprecations: Anthropic retires old models with notice; pinned dated IDs make migrations explicit. **Re-run evals when switching models — prompts do not transfer performance automatically.**
- Cost/usage monitoring: `usage` block per response; batch vs real-time mix; cache hit rates (`cache_read_input_tokens`).

1.10 · Tech fundamentals

`~3.5–4 chars`	`~750 words`	`count_tokens`	`temp 0.0 → 1.0`
≈ 1 TOKEN (ENGLISH)	≈ 1K TOKENS	FREE ENDPOINT, OWN LIMITS	DETERMINISTIC → CREATIVE

- Everything is billed in tokens: system prompt, messages, tool definitions, tool results, thinking, output.
- **Context window** = ALL input + output for a request: system + history + tool defs + tool results + thinking + response.
- **Token counting endpoint:** `POST /v1/messages/count_tokens` — **free**, separate rate limits, accepts messages/system/tools/images; returns an estimate.
- Temperature: 0.0 → most deterministic (classification/extraction), 1.0 (default) → creative. **Don't set both `temperature` and `top_p`.**
- SSE = Server-Sent Events (the streaming transport). JSON Schema = the contract language for tools and structured output.

IN THE FULL EDITION — SIX MORE SECTIONS OF THIS DEPTH

- Domain 2 · Model Selection and Optimization
- Domain 3 · Agents and Workflows
- Domain 4 · Prompt and Context Engineering
- Domain 5 · Tools and MCPs
- Domain 6 · Security and Safety
- Domain 7 · Claude Code
- Domain 8 · Eval, Testing, and Debugging
- Memorize-me quick reference

Practice bank — 36 questions

36 original, exam-style practice questions with full explanations — written to the published objectives, not taken from any exam.

Work them by domain as checkpoints (the 7-day plan says when), then re-run your misses on day 7. The answer key starts on its own page after the last question — answers follow on the next page.

Applications and Integration

Q1 EASY

Your production endpoint starts returning HTTP 429 with error type rate_limit_error during a traffic spike. What is the correct client-side behavior?

A. Rotate to a backup API key and immediately resend the failed request

B. Wait the number of seconds indicated by the retry-after header, then retry with exponential backoff on subsequent failures

C. Treat it as a permanent authentication failure and alert on-call

D. Switch the request to stream: true, since streamed requests are not rate limited

Q2 EASY

You need 10,000 product descriptions summarized by tomorrow morning at the lowest possible cost. No user is waiting on individual results. Which approach is best?

A. Send 10,000 parallel real-time requests to Claude Opus 5 with streaming enabled

B. Send 10,000 sequential real-time requests to Claude Sonnet 5

C. Submit one Message Batches API job using Claude Haiku 4.5

D. Submit one Message Batches API job using Claude Opus 5 with fast mode enabled

Q3 EASY

A response comes back with stop_reason: "max_tokens". What does this tell you?

A. The model hit your max_tokens limit and the response is truncated

B. The request exceeded the model's context window and was rejected

C. The model finished its answer naturally within the token budget

D. Your account hit its monthly token quota

Q4 MEDIUM

After Claude responds with a tool_use block, how do you return the tool's output to the API?

A. As a message with role "tool" containing the raw output string

B. As a user-role message containing a tool_result content block that references the tool_use_id

C. As an assistant-role message containing a tool_result content block

D. By calling a dedicated /v1/tool_results endpoint with the tool call id

Q5 MEDIUM

While streaming a response in which Claude calls a tool, which SSE delta type delivers the tool's arguments?

A. text_delta events containing stringified JSON

B. tool_use_delta events containing complete argument objects

C. input_json_delta events containing partial JSON string fragments

D. message_delta events containing the accumulated input object

Answer key & explanations

Read every explanation, even for questions you got right — the distractor rationales are training too.

Q1 → B 429 means a rate limit was hit; the response includes a retry-after header telling you how long to wait, and retries should use exponential backoff. Rotating keys to evade limits violates the intent of per-organization limits and doesn't fix bursty clients. It is not an auth error (that's 401). Streaming does not exempt requests from RPM/ITPM/OTPM limits.

Q2 → C Batch processing gives a 50% token discount with a 24-hour processing window (most batches finish within an hour), which comfortably meets an overnight deadline; Haiku 4.5 is the cheapest model and sufficient for summarization. Real-time Opus with streaming (A) is the most expensive option and streaming saves no money. Sequential Sonnet (B) forgoes the 50% batch discount. Fast mode (D) is not supported on the Batch API at all.

Q3 → A stop_reason: "max_tokens" means generation stopped because it reached the max_tokens value you set, so the output is likely cut off — raise the limit or continue the response. A rejected over-limit request is an HTTP error (or model_context_window_exceeded), not max_tokens. Natural completion is end_turn. Quota/spend issues surface as 429 errors, not stop reasons.

Q4 → B Tool results go back in the next user message as a tool_result content block whose tool_use_id matches the id from the tool_use block. The Messages API has no "tool" role (B's key distinction from OpenAI-style APIs). Assistant messages carry the model's own output, including the original tool_use block, not results. There is no separate tool-results endpoint — everything flows through /v1/messages.

Q5 → C Tool arguments stream as input_json_delta events, each carrying a partial_json string fragment that you accumulate and parse when the content block stops. text_delta carries ordinary text content. There is no tool_use_delta event type. message_delta carries top-level message changes (like stop_reason and usage), not tool arguments.

Ready for the rest?

You've just read about 32% of the full guide — at full depth, nothing dumbed down.

The full Unofficial CCDV-F Study Guide includes

- Every domain at the same depth as Domain 1 you just read — diagrams, decision tables, trap pairs, and cue boxes throughout
- The complete 7-day plan: 7 days of guided study, hands-on drills, and checkpoints
- All 36 original practice questions with full answer-key explanations — including the distractor rationales
- The quick-reference sheets and night-before cram material

Get the full edition at `zehon.com`