INDEPENDENT EXAM PREPARATION · CCA-F

# Claude Certified Architect

## *Foundations · Complete Study Programme*

A foundations primer, five full-depth theory chapters each with extended case studies, six scenario playbooks, ten hands-on labs with solutions, five 50-question domain tests with fully worked keys, a distractor compendium, 150 flashcards, revision plans, and a complete glossary. Five mock examinations ship in the companion booklet, and a web test engine runs the full bank interactively.

250 test questions, worked keys | 6 scenario playbooks | 10 labs | 150 flashcards | 720 / 1000 to pass

PART 1

# Orientation

What the exam is, how it is scored, how its questions work, and how to use this manual and its companion test engine to prepare in the fewest wasted hours.

# The exam at a glance

| | |
|---|---|
| **Credential** | Claude Certified Architect – Foundations (CCA-F), Anthropic's foundations-tier technical certification, launched 12 March 2026. "Foundations" is the entry tier of a planned multi-level credential stack, with further tiers announced for sellers, developers and advanced architects. |
| **Format** | 60 scenario-based multiple-choice questions, each with one correct answer and three distractors, in a single proctored session of approximately 120 minutes. No external resources, no breaks. |
| **Scenario draw** | Questions are anchored to six published scenario contexts; four of the six are drawn at random for any given sitting. |
| **Scoring** | Scaled score on a 100–1,000 range; the pass mark is 720. A widely circulated "850" figure is a practice-buffer suggestion, not the bar. |
| **Level and audience** | An exam aimed at solution architects with roughly six months of hands-on experience across the Claude API, the Claude Agent SDK, Claude Code and MCP. |
| **Cost and access** | Standard fee 99 USD at the time of writing: verify the current fee at booking. Registration, the practice exam and the free preparation courses are hosted on Anthropic's Skilljar platform (anthropic.skilljar.com). |

Table sorted by weight, not domain number.

## The blueprint: five weighted domains

Preparation time should track the weights. Domain 1 alone carries more than a quarter of the marks; together with Domains 3 and 4 it accounts for two thirds of the exam.

| DOMAIN | WEIGHT | WHAT IT TESTS |
|---|---|---|

| | | |
|---|---|---|
| 1 · Agentic Architecture & Orchestration | 27% | Agentic loops and termination, single versus multi-agent design, coordinator and subagent patterns, task decomposition, context delegation, session state, graceful partial failure, multi-pass review |
| 3 · Claude Code Configuration & Workflows | 20% | The `CLAUDE.md` hierarchy, path-scoped rules, skills, slash commands, hooks, plan mode, built-in tools, headless use in CI/CD |
| 4 · Prompt Engineering & Structured Output | 20% | Explicit criteria, few-shot design, prompt structure, structured JSON via tool use, validation and retry, batching, model selection, multi-pass review prompting |
| 2 · Tool Design & MCP Integration | 18% | Tool descriptions and selection, schema design, structured tool errors, distributing tools across agents, MCP primitives, servers, transports and wiring |
| 5 · Context Management & Reliability | 15% | Context window management, compaction, scratchpads, prompt caching, large-codebase exploration, error propagation, retries, escalation thresholds |

## Anatomy of a scenario question

Every item on this exam has the same skeleton: a system described in two or three sentences, a constraint or a failure, and four designs. Exactly one design is correct *for the stated constraint*; the other three are plausible engineering that breaks under it. Dissect one example completely, because the habit of reading questions this way is worth more than any single fact in this book.

**Worked example** DOMAIN 1

A research assistant uses a coordinator that delegates to three subagents: search, analysis and writing. Each subagent receives the full conversation history. Token costs are rising and the writing subagent has begun ignoring late instructions. What should the architect change first?

A Move to a model with a larger context window so the history fits comfortably.

B Pass each subagent a scoped brief for its subtask and have it return a condensed summary.

C Add a fourth subagent to summarise the history before each delegation.

D Cache the conversation history so resending it costs less.

PART 2

# Foundations primer

The mechanics underneath every domain: how a request is shaped, how tokens and windows behave, how a tool call physically round-trips, and the vocabulary the exam assumes. Experienced builders can skim; newcomers should read this before Domain 1.

# The request, mechanically

Everything in this programme ultimately becomes one HTTP call to the Messages API. The request names a model, sets a hard output budget (`max_tokens`), optionally supplies a `system` prompt, and carries a `messages` array of alternating `user` and `assistant` turns. The response comes back as a list of content blocks plus a `stop_reason` that says why generation ended. That is the whole physical substrate; agents, tools, and orchestration are disciplined arrangements of this one call.

```
import anthropic
client = anthropic.Anthropic()

resp = client.messages.create(
    model="claude-sonnet-4-5",
    max_tokens=1024,
    system="You are a precise billing assistant. Answer from the supplied policy
only.",
    messages=[
        {"role": "user", "content": "A customer was charged twice for order 8841. What
do we do?"}
    ],
)
print(resp.stop_reason)          # e.g. "end_turn"
print(resp.content[0].text)      # the assistant's reply
```

Four properties of this call matter for everything that follows:

- **The call is stateless.** The model remembers nothing between requests. Continuity exists only because your code re-sends the conversation each time. Every "session", "memory", or "agent" in this manual is state your side of the wire.
- **The system prompt is the durable frame.** Role, constraints, policy, and output contract go here; per-request material goes in the user turn. This split is also what makes the frame cacheable (Domain 5).
- **`max_tokens` is a budget, not a target.** Generation stops at `end_turn` when the model finishes, or at `max_tokens` when the budget runs out mid-thought. A `stop_reason` of `max_tokens` on a truncated answer is a configuration symptom, not a model failure.
- **Turns alternate.** The messages array is a strict user/assistant alternation. Tool results, as you will see, ride in `user` turns; that is not a quirk but the protocol.

# Tokens and windows

Models read and write tokens, not characters. In ordinary English a token averages roughly three to four characters, so a thousand words is on the order of 1,300–1,500 tokens; code, JSON, and unusual strings tokenise less efficiently. Two budgets are always in play:

| | |
|---|---|
| **The context window** | The total the model can attend to in one call: system prompt, every conversation turn, every tool definition, every tool result, and the response being generated, all counted together. Modern Claude windows run to hundreds of thousands of tokens, which feels unlimited until an agent starts accumulating tool output, at which point Domain 5's disciplines stop being optional. |
| **The output budget** | `max_tokens` caps the response alone. It protects you from runaway generation and protects latency; set it to what the task's answer actually needs. |

The exam never asks you to count tokens. It asks whether you understand the consequences: that everything in the conversation is paid for on every call, that attention dilutes across crowded windows before capacity runs out, and that "use a bigger window" postpones rather than fixes a relevance problem.

# A tool call, end to end

Tool use is the mechanism that turns a text model into an actor, and its round trip is the single most load-bearing mechanic in the exam. Walk it once, slowly.

**Step 1: you declare tools.** Each tool is a name, a description, and a JSON Schema for its input. The model never sees your implementation; this interface is everything it knows.

```
tools = [{
    "name": "get_order",
    "description": ("Look up one order by its internal ID and return status, items, "
                    "and charge history. Use for questions about a specific known
order. "
                    "For searching across orders, use search_orders instead."),
    "input_schema": {
        "type": "object",
        "properties": {
            "order_id": {"type": "string", "description": "Internal order ID, e.g.
ORD-8841"}
        },
        "required": ["order_id"],
    },
}]
```

**Step 2: the model elects to call.** If the model decides a tool is needed, the response arrives with `stop_reason: "tool_use"` and a `tool_use` content block carrying a generated `id`, the tool `name`, and an `input` object shaped by your schema. Nothing has executed; the model has asked.

**Step 3: your runner executes.** Your code, the runner, finds the named function, runs it with the supplied input, and captures a result or a structured error. The model is not running your code; you are, which is why hooks and permission gates (Domain 3) can sit between the ask and the act.

**Step 4: results return in a user turn.** You append the assistant's tool-use turn to the history, then a `user` turn containing a `tool_result` block whose `tool_use_id` matches the request, and call the model again. The matching ID is what lets the model pair answers to questions when it issued several calls at once.

```
follow_up = {
    "role": "user",
    "content": [{
        "type": "tool_result",
        "tool_use_id": tool_block.id,
        "content": '{"order_id":"ORD-8841","status":"delivered","charges":2}',
    }],
}
```

**Step 5: the loop continues until a stop.** The model may answer, or ask for another tool, and the runner repeats. The published `stop_reason` values you must read fluently: `end_turn` (the model believes it has finished), `tool_use` (it awaits results), `max_tokens` (budget exhausted mid-generation), and `stop_sequence` (one of your declared stop

strings was emitted). Domain 1 builds its whole termination doctrine on top of these signals.

## Sampling, briefly

Temperature controls how adventurously the model picks among likely next tokens: low values produce consistent, repeatable phrasing; higher values vary it. The production rule the exam rewards is unglamorous: anything parsed, validated, or routed downstream runs cold; deliberate variety (brainstorming, drafting alternatives) is the only reason to warm it up. Temperature is also the exam's favourite wrong answer: it cannot create policy, enforce schemas, prevent invented values, or substitute for any structural mechanism, and options that reach for it usually mark a distractor.

## The model tiers

Claude ships in three production tiers, and matching tier to step is a scored skill (Domain 4):

| TIER | PROFILE | NATURAL WORK |
|---|---|---|
| Haiku | Fastest, cheapest | Routing, triage, classification, simple extraction at volume |
| Sonnet | The production workhorse | Default for most real work: agents, coding, drafting, analysis |
| Opus | Deepest reasoning, highest cost | The genuinely hard remainder: subtle analysis where errors are expensive and volume is low |

The two named anti-patterns: the biggest model everywhere (fails the cost constraint) and the cheapest everywhere (fails the quality constraint). Composite designs, a cheap tier triaging in front of a workhorse, with the top tier reserved for the hard tail, recur throughout the question bank.

## The exam's working vocabulary

Scenario stems use this vocabulary without glossing it. If any row is unfamiliar, the referenced chapter is where it is built properly; the full alphabetical glossary sits in Part 8.

| TERM | ONE-LINE MEANING | BUILT IN |
|---|---|---|
| Agentic loop | Model turn, tool execution, results appended, repeat until a terminal condition | D1 |
| Terminal condition | A designed reason for the loop to end: validated completion, handoff, unrecoverable error | D1 |
| Runner | Your code around the model: executes tools, feeds results back, enforces stops | D1 |
| Subagent | A separately framed, separately privileged agent spawned for scoped work | D1, D3 |
| Scoped brief | What a delegation carries in: the question, relevant facts, constraints, output contract | D1 |
| Structured error | A tool failure carrying category, retryability, and an actionable message | D2 |
| MCP host / server | The AI application consuming capabilities / the process exposing them over the protocol | D2 |
| Hook | Deterministic code at a lifecycle event, enforced by the runtime rather than requested of the model | D3 |
| Plan mode | Research and propose read-only, then execute after approval | D3 |
| Headless / print mode | `claude -p`: one task, structured output, exit; the CI invocation | D3 |
| Forcing structure | Using a tool schema plus `tool_choice` so output arrives schema-shaped | D4 |
| Abstention | The designed ability to answer "not present": nullable fields plus the instruction to use them | D4 |
| Compaction | Replacing older turns with a summary that preserves decisions, facts, and open threads | D5 |
| Scratchpad | State moved outside the window into a durable store the agent reads and writes | D5 |

| | | |
|---|---|---|
| Prompt caching | Reusing a byte-stable prefix across calls for cost and latency | D5 |
| Graceful degradation | Reduced but honest service when a dependency fails, with the limitation stated | D5 |

HOW THE PRIMER CONNECTS FORWARD

Domain 1 turns the single call into a loop with judgement. Domain 2 designs the tool interfaces the loop selects among. Domain 3 applies both inside Claude Code, where the runner, the tools, and the configuration surfaces are product features. Domain 4 shapes what goes into the call and what comes out of it. Domain 5 keeps the whole arrangement affordable, attentive, and honest over time. Every one of the 550 questions in this programme resolves to a decision about one of those five layers sitting on the mechanics you have just read.

END OF PREVIEW

# What the rest of it looks like

Those were ten pages out of 298, reproduced unaltered and carrying their own page numbers: the orientation pages 5 to 7, then the whole of the foundations primer, pages 13 to 19. The primer is where the mechanics get settled. How a request is actually built, what the model sees, what stop_reason means, and why the agentic loop is the concept the exam keeps returning to. Everything after it assumes you have read it.

IN THE STUDY MANUAL

| | |
|---|---|
| **298 pages** | A foundations primer, then one full-depth chapter per domain, each with extended case studies. |
| **250 questions** | Five 50-question domain tests. Every key explains the correct option and why each of the three distractors fails. |
| **6 playbooks** | The six published scenario contexts worked end to end, with failure drills. |
| **10 labs** | Hands-on exercises with solutions, common stumbles and extensions, plus ten annotated build recipes. |
| **12 archetypes** | The distractor compendium, with twenty worked eliminations and two cross-domain walkthroughs. |
| **150 flashcards** | Prompt and answer, grouped by domain, plus ninety rapid-recall drills and a full glossary. |

AND IN THE REST OF THE SYSTEM

- A 153-page mock examination booklet: five full papers at the real domain weights, 300 further questions, every one worked.
- A web test engine running all 550 questions in study and exam modes, on a 120-minute clock, scored against the 720 band.
- Two study plans built into the manual: roughly twenty hours over two weeks for experienced builders, or forty over four alongside Anthropic Academy's free courses.

## One purchase. The whole system.

Both books, the engine and the flashcards, delivered digitally the moment you buy.
ccafoundations.com