



Vendor: Anthropic

Exam Code: CCDV-F

Exam Name: Claude Certified Developer - Foundations

Version: DEMO

QUESTION 1

A teammate is reviewing the team's threat model for a Claude application and has asked you to identify the categories of AI-specific threats that the model should cover. The teammate has already listed traditional web application threats and wants to know what additional categories apply to a Claude application. Which AI-specific threat categories would you add?

- A. Cross-site scripting and SQL injection, because these traditional web application threats apply with equal weight to any application that uses Claude in any way.
- B. Network-level denial of service and physical infrastructure attacks, because these categories cover the threats most likely to affect any Claude application in production.
- C. Prompt injection, data leakage from prompts or context, jailbreak attempts, and unsafe model output that bypasses application controls.
- D. Supply chain attacks on the Claude SDK because the SDK itself is the only point of vulnerability that a Claude application introduces beyond traditional web application threats.

Answer: C

Explanation:

Option C correctly identifies threat categories introduced or significantly amplified by LLM-based application architecture. Prompt injection attempts to manipulate Claude into following adversarial instructions. Jailbreaks seek to circumvent behavioral or application safeguards. Data leakage can expose confidential information contained in system prompts, retrieved context, tool results, or conversation state. Unsafe model output becomes particularly serious when output is consumed by downstream systems or translated into tool actions.

Anthropic's official guardrail documentation explicitly distinguishes jailbreaks and direct prompt injection from indirect prompt injection. It recommends input screening, hardened system prompts, structured handling of untrusted tool content, least-privilege access, output screening, and continuous monitoring. Anthropic also warns that sensitive prompt/context data can leak and recommends post-processing and output filtering where required.

QUESTION 2

Your Claude application uses structured output that is consumed by downstream code. The team wants to handle malformed or unexpected output gracefully so it does not crash downstream systems. The best choice for handling this issue would be to...

- A. Crash on any unexpected output and require manual recovery on affected requests the application handles during normal operation.
- B. Silently drop any output that does not match expectations so downstream systems do not see the malformed output the application receives from the model.
- C. Use the output the model produces and leave schema enforcement to a later phase of production after the application is more complete.
- D. Apply defensive parsing to handle unexpected fields, missing values, and type mismatches that the downstream code might otherwise crash on.

Answer: D

Explanation:

Option D applies a fundamental production engineering principle: treat externally generated data as potentially malformed and parse it defensively before use. Downstream application logic should not assume that every field exists, every type is correct, or every unexpected property can safely be ignored. Instead, the parser should validate expected structures, handle optional or missing values deliberately, reject incompatible types, and convert failures into controlled application errors rather than process crashes.

Anthropic's Structured Outputs documentation identifies exactly these failure classes for

unconstrained model output: malformed JSON, missing required fields, inconsistent types, and schema violations can break downstream applications. Current Structured Outputs and strict tool-use capabilities can eliminate many schema-level failures through constrained decoding, but defensive handling remains an important software boundary when unexpected data can still arise from external services, legacy paths, or semantic validation requirements.

QUESTION 3

You are designing a Claude application that will process customer support tickets in two stages: a triage stage that classifies tickets and a response stage that drafts replies. The team is debating whether to use a single Claude call that handles both stages or separate Claude calls for each stage. How would you structure the application?

- A. Use a single Claude call for triage and then use a non-Claude rule-based system for response generation, on the grounds that rule-based systems are more reliable for drafting replies.
- B. Use multiple Claude calls in parallel that each draft a complete ticket reply, then have a fourth Claude call select the best one to send to the customer.
- C. Use separate Claude calls for triage and response, because each stage has distinct inputs, outputs, and success criteria that benefit from focused prompts.
- D. Use a single Claude call for both stages, on the grounds that a single call is cheaper than multiple calls in any production Claude application setup.

Answer: C

Explanation:

Option C follows a prompt-chaining/workflow decomposition pattern. Ticket classification and reply generation are materially different tasks. Triage may require a constrained category, priority, or routing decision, while response generation requires richer context, tone, policy compliance, and customer-facing language. Splitting the stages allows each Claude call to use a prompt, context set, model configuration, output format, and evaluation criterion appropriate to that specific responsibility.

Anthropic's guidance on effective agentic systems distinguishes predictable workflows from open-ended agents and recommends using the simplest architecture that meets requirements. Predefined workflows are particularly appropriate where work can be divided into clear processing stages.

This decomposition also creates useful engineering boundaries: the triage output can be validated before becoming input to the drafting stage; triage accuracy and response quality can be evaluated independently; and either stage can later be optimized without rewriting the entire process.

QUESTION 4

You are reviewing an architectural diagram for a Claude-powered travel-booking system. The diagram shows a top-level component that interprets user requests and three subordinate components that handle flights, hotels, and ground transportation. The top-level component is responsible for routing each request, sequencing the subordinate components, and reconciling their outputs into a final itinerary. The diagram also shows that each subordinate component has its own tool list and own short conversation history that is not shared with the others. Which architectural pattern does this diagram most closely describe?

- A. A manager and supervisor pattern with isolated context per subagent, where the top-level agent coordinates specialized subagents that each maintain their own conversation history and tool list.
- B. A manager and supervisor pattern with shared context, where the top-level agent and the subagents all share a single conversation history that grows as the request flows through the

system.

- C. A pipeline pattern, where each component processes the user request in sequence and passes the full conversation history along with the request to the next component in the pipeline.
- D. A retrieval-augmented pattern, where the top-level component retrieves relevant context from the subordinate components' indexed data stores before generating each response.

Answer: A

Explanation:

The architecture is a manager/supervisor--or orchestrator/subagent--pattern with isolated subagent context. The defining characteristics are a coordinating top-level agent, specialized subordinate agents, delegated responsibilities, and distinct tool/context boundaries. The supervisor determines which specialist should act, sequences work where necessary, and integrates the specialists' outputs into the final result.

Anthropic's published multi-agent research architecture uses this same structural principle. Its Research system employs an orchestrator-worker pattern in which a lead agent analyzes the task, creates specialized subagents, delegates separate research responsibilities, and consolidates their findings. Anthropic emphasizes that good delegation requires each subagent to receive a specific objective, task boundaries, expected output, and appropriate tools.

QUESTION 5

Your Claude application's API keys are stored in a secrets manager. The team is debating whether the same key should be used in development, staging, and production environments. How would you handle the keys across environments?

- A. Use distinct keys for each environment so a compromise in one environment does not affect the others during normal operation across the application's lifecycle.
- B. Rotate the same key across environments at random intervals on the grounds that random rotation provides isolation between environments without requiring distinct keys.
- C. Use the same key across all environments for simplicity and treat the propagation of any compromise as a known operational tradeoff for the team's key management approach.
- D. Use a single development key everywhere on the grounds that production keys are too risky to deploy across the application's three environments during normal operation.

Answer: A

Explanation:

Option A provides proper environment isolation and credential blast-radius control. Development, staging, and production represent different trust boundaries and should not share the same API credential. If a development machine, CI job, or staging service is compromised, a distinct credential prevents the attacker from automatically gaining the same access to production.

Anthropic's official workspace documentation explicitly recommends using separate workspaces for development, staging, and production. Workspaces can have their own API keys, members, resource limits, and usage tracking, and keys can be scoped to a specific workspace. Anthropic's authentication guidance also instructs developers to store API keys in a secrets manager, rotate them periodically, revoke suspected compromised keys, and use workspaces to scope credentials by project or environment.

QUESTION 6

A teammate has asked you to explain why the team's Claude application is billed for output tokens at a different rate than input tokens. They had assumed the rate was the same for both. How would you explain the difference?

- A. Output tokens are typically billed at the same rate as input tokens, and the apparent rate difference is a billing error to report to Anthropic.
- B. Output tokens are typically billed at a lower rate than input tokens, because output tokens are cheaper to produce than input tokens are to process.
- C. Output tokens are not billed at all, because cost is determined entirely by the input tokens sent to the model on each request.
- D. Output tokens are typically billed at a higher rate than input tokens, and cost models for the application should reflect both rates separately.

Answer: D

Explanation:

The supplied Claude Certified Developer Foundations question identifies D as the correct answer. Anthropic prices input and output tokens separately, so application cost models must independently account for the number of tokens supplied to Claude and the number generated in response.

Anthropic's official pricing documentation confirms that output-token pricing is normally higher than base input-token pricing. For example, current Claude models maintain distinct columns for base input tokens and output tokens; the exact dollar amounts depend on the selected model. This distinction matters because an application with relatively small prompts but very large generated responses can incur substantial output-token expenditure even when input volume is modest.

QUESTION 7

A Claude application is occasionally refusing to answer questions that should be in scope, including questions the application has answered correctly in the past. You want to investigate. What is the first step of your investigation?

- A. Test whether removing or adjusting those instructions resolves the behavior.
- B. Examine traces of the refused requests to identify what triggers the refusal, whether input patterns, system prompt content, or other context.
- C. Assess whether the refusals cluster around specific input patterns, question types, or time periods by analyzing the distribution of refused requests in the logs.
- D. Identify which questions have been refused and compare them against previously answered questions to determine what changed between the successful and refused interactions.

Answer: B

Explanation:

The supplied exam source marks B as correct. The first debugging action should be evidence collection rather than immediately modifying prompts or configuration. A trace exposes the complete conditions under which the refusal occurred: user input, system instructions, accumulated conversation state, tool results, parameters, and resulting model output.

This is important because a refusal can originate from several interacting causes. A particular input pattern may trigger it; system instructions may unintentionally conflict with the requested task; prior conversational context may alter Claude's interpretation; or contextual information may make an otherwise acceptable request appear out of scope. Without examining the failing execution path, changing instructions is speculative.

QUESTION 8

You are starting a new Claude application and have a small set of well-labeled examples that demonstrate the desired output format. You want to use these examples to guide Claude's behavior. How would you guide the application's behavior?

- A. Use multi-shot prompting by including the labeled examples in the prompt so Claude can match the desired output format on each request.
- B. Use zero-shot prompting and rely on the model's general capability to produce the desired output format consistently across all incoming requests.
- C. Embed the examples in a database and retrieve them at runtime as reference material for the team.
- D. Train a custom model on the labeled examples before deployment so that the application does not need to include the examples in any prompt during operation.

Answer: A

Explanation:

The supplied examination set identifies A as correct. A small collection of high-quality labeled examples is ideally suited to few-shot or multishot prompting. The examples demonstrate concretely what acceptable input/output behavior looks like, allowing Claude to infer formatting, structure, tone, and task-specific conventions without requiring model retraining.

Anthropic's official prompting guidance states that examples are among the most reliable ways to steer Claude's output format, tone, and structure. It recommends using relevant, diverse examples and clearly separating them from the surrounding instructions. Anthropic currently recommends approximately three to five examples where practical and suggests XML structures such as `<examples>` and `<example>` to make prompt organization explicit.

QUESTION 9

A teammate has asked you to explain when a Skill would be the right choice over an MCP server. The teammate is unsure how the two differ in practice when both can be reused across teams. How would you explain the distinction?

- A. A Skill and an MCP server are equivalent extension mechanisms that the team can use interchangeably for any reusable capability that needs to be accessible across teams.
- B. A Skill is the older mechanism and an MCP server is the newer one, so the team should prefer an MCP server for any reusable capability that the team builds going forward.
- C. A Skill is preferable for cross-team reuse because it loads more efficiently than an MCP server during normal operation in the team's typical multi-team workloads.
- D. A Skill bundles prompts, scripts, and data into a package the model loads as a unit while an MCP server exposes resources, tools, and prompts through a standard client interface.

Answer: D

Explanation:

The supplied Claude Developer question explicitly identifies D. The two mechanisms solve different extension problems. A Skill is a reusable package of domain expertise and workflow guidance. Anthropic describes Agent Skills as modular, filesystem-based capabilities containing instructions, metadata, and optional supporting resources such as scripts and templates, which Claude can load when relevant.

MCP, by contrast, defines a standardized mechanism for exposing external capabilities and context. Anthropic's MCP integration supports MCP tools and, through client-side helpers, MCP prompts and resources. This makes MCP appropriate when Claude must communicate with an external server or reusable service interface rather than simply load packaged expertise.

Thus, Skills are appropriate for packaging repeatable instructions, procedures, scripts, templates, or domain knowledge. MCP servers are appropriate for exposing callable operations, external data resources, and shared service capabilities using the Model Context Protocol.

QUESTION 10

Your Claude agent has access to a tool that retrieves customer records. A teammate has noticed that the agent occasionally calls the tool with arguments the schema does not declare, and the tool's downstream service returns an error each time. The teammate proposes loosening the schema so the tool accepts whatever arguments the model produces. How would you respond?

- A. Add a system prompt instruction telling the model to produce schema-conforming arguments, treating the prompt instruction as the primary mechanism for keeping the agent's tool calls valid.
- B. Keep the schema strict, validate arguments before dispatching, and return a structured error so the agent can retry.
- C. Remove the schema entirely and rely on the downstream service to reject invalid calls, treating the downstream service as the team's primary enforcement layer.
- D. Loosen the schema as the teammate proposed so the downstream service receives every call the agent makes during normal operation.

Answer: B

Explanation:

The supplied question identifies B as the intended answer. Tool schemas are contracts between Claude and executable application code. When the downstream service accepts only a defined set of arguments, relaxing that schema merely shifts invalid data farther into the system and increases runtime failures.

Anthropic's current tooling provides an even stronger implementation of this principle through strict tool use. Setting `strict: true` constrains tool inputs to the declared JSON Schema, preventing undeclared properties, missing required values, and incompatible parameter types where the supported schema subset is used. Anthropic explicitly recommends strict tool use for validated parameters, type-safe function calls, and reliable agentic workflows.

QUESTION 11

You are designing a Claude application that maintains user sessions across multi-turn conversations. The product team has asked how the application will handle session lifecycle: when sessions should expire, how state is reset, and how the application avoids carrying stale context into new conversations. How would you design session lifecycle?

- A. Define explicit session expiration rules, state reset triggers, and rules for starting fresh sessions so stale context does not leak into new conversations.
- B. Define a single short session timeout that applies across all conversations and treat the timeout as the application's complete session lifecycle mechanism.
- C. Define explicit session expiration rules but rely on users to start new conversations when they want fresh context, with no automatic reset triggers in the application.
- D. Define state reset triggers tied to specific application events but apply them across all sessions globally, with no per-session expiration rules.

Answer: A

Explanation:

A is the correct lifecycle architecture. The supplied Claude Certified Developer Foundations item explicitly selects A. A session is a state boundary, so its lifecycle must define when accumulated conversation state remains valid and when that state must be discarded. Anthropic's Managed Agents documentation establishes that a session maintains conversation history across multiple interactions and provides explicit session operations including creation, update, archive, and deletion.

A therefore addresses all three required controls: expiration, reset, and fresh-session creation.

Expiration prevents indefinitely retained conversational state. Reset triggers allow the application to clear state when events such as logout, workflow completion, tenant switching, or explicit user reset occur. Fresh-session rules prevent context belonging to an earlier task from influencing a logically unrelated conversation.

QUESTION 12

A team has deployed a multi-agent system in which a primary agent decomposes user requests and delegates subtasks to three specialized subagents: one for data retrieval, one for analysis, and one for report generation. In production, the team observes that subagents are making redundant tool calls, occasionally exceeding token budgets, and sometimes producing outputs that contradict each other -- all of which the primary agent passes along without catching. What is the most appropriate way to address these failures?

- A. Add retry logic to each subagent so that when a tool call fails, the subagent retries automatically before escalating -- and configure each subagent to log its tool calls and outputs to a shared trace so the team can audit redundancy and contradictions after the fact.
- B. Give each subagent read access to the other subagents' outputs so they can identify and resolve contradictions without routing back through the primary agent.
- C. Strengthen the primary agent's management layer to enforce per-subagent tool budgets, validate outputs against a defined schema before passing them forward, and establish explicit handoff contracts between stages.
- D. Collapse the three subagents into a single large-context model call that handles retrieval, analysis, and generation in one pass.

Answer: C

Explanation:

C addresses the failures at the correct architectural layer: orchestration and supervision. The supplied exam item identifies C as correct. The primary agent is responsible not merely for forwarding subordinate output but for governing delegation, resource usage, stage boundaries, and result quality.

Anthropic's current guidance recommends explicit control over subagent use because excessive delegation multiplies latency and cost. Its documentation specifically supports deterministic limits on subagent spawning and SDK budget controls such as `max_budget_usd`. Tool contracts can also use defined input schemas and strict validation so malformed data cannot silently propagate between processing stages.

Explicit handoff contracts are equally important. Retrieval should provide an agreed structure to analysis; analysis should provide validated findings to report generation; and the manager should reject, retry, reconcile, or escalate outputs that violate those contracts.

Thank You for Trying Our Product

Lead2pass Certification Exam Features:

- ★ More than **99,900** Satisfied Customers Worldwide.
- ★ Average **99.9%** Success Rate.
- ★ **Free Update** to match latest and real exam scenarios.
- ★ **Instant Download** Access! No Setup required.
- ★ Questions & Answers are downloadable in **PDF** format and **VCE** test engine format.
- ★ Multi-Platform capabilities - **Windows, Laptop, Mac, Android, iPhone, iPod, iPad**.
- ★ **100%** Guaranteed Success or **100%** Money Back Guarantee.
- ★ **Fast**, helpful support **24x7**.



View list of all certification exams: <http://www.lead2pass.com/all-products.html>



10% Discount Coupon Code: ASTR14