PassQueen
a.k.a dumpsqueen.com

Claude Certified Developer-Foundations

Anthropic CCDV-F

Version Demo

Total Demo Questions: 11

Total Premium Questions: 111

Buy Premium PDF

https://passqueen.com

support@passqueen.com

QUESTION NO: 1

Your Claude application has multi-step workflows where each stepâ€™s output is needed only briefly before the agent moves on. The cumulative tool output is filling the context window with content that is no longer relevant.

How would you handle the accumulating tool output?

A. Apply tool output pruning to remove tool outputs that are no longer needed by later steps in the workflow.

B. Apply prompt caching to the accumulated tool outputs so the application does not re-pay for the older content on each subsequent step.

C. Switch to a smaller Claude model that processes context more efficiently and treat any quality loss as a tradeoff for the cost reduction.

D. Keep every tool output in the context indefinitely so the agent has the full record of every step it has executed during the workflow.

ANSWER: A

Explanation:

Apply tool output pruning to remove tool outputs that are no longer needed by later steps in the workflow. This is the appropriate context-engineering approach when an agent produces temporary search results, file contents, API responses, or intermediate artifacts that have served their purpose. Removing stale results preserves space in the finite context window for the current task, relevant recent interactions, and durable state the agent still needs. It also reduces the amount of low-signal information the model must consider, which can improve focus and limit unnecessary token use in multi-step workflows.

Anthropic supports this pattern through context editing, including tool-result clearing. An application can configure Claude to clear older tool results as the conversation grows, while retaining recent results or specific tool outputs that remain necessary. The workflow should therefore keep concise, high-value summaries or structured state when needed, then discard the bulky raw outputs once Claude has extracted the information required for subsequent steps. This produces a deliberately curated context rather than treating the transcript as permanent storage. See Anthropic's Context Editing documentation and its prompt-engineering guidance.

QUESTION NO: 2

You are designing an agent that processes vendor invoices. The work involves a small number of well-understood steps, but occasionally an invoice arrives in an unexpected format that requires the system to decide between rerouting, requesting clarification, or flagging for human review.

The most appropriate architecture for this system is...

A. A fully autonomous agent that handles every invoice from start to finish across all formats.

B. A manager agent that delegates each step of standard invoice processing to a dedicated subagent, with a separate subagent handling each unexpected format.

C. A single large prompt that processes every incoming invoice, both standard and unexpected, in one model call.

D. A workflow for the standard path with an agent invoked at the decision point for unexpected formats.

ANSWER: D

Explanation:

A workflow for the standard path with an agent invoked at the decision point for unexpected formats is correct because it applies the simplest effective architecture to each part of the process. Standard invoice handling consists of a small set of

known, repeatable operations, so a workflow can explicitly define the sequence, validations, tool calls, and expected outputs. This makes the normal path predictable, testable, observable, and easier to operate reliably at scale.

The unexpected-format condition introduces a genuine judgment point. The system may need to inspect the available information and choose an appropriate response, such as rerouting the invoice, asking for clarification, or escalating it to a human. Invoking an agent specifically at that bounded exception point provides the flexibility for model-driven reasoning without making routine processing unnecessarily autonomous. The workflow can also retain control of the overall process by defining guardrails, approval requirements, and the conditions under which the agent is called.

This design follows Anthropic's guidance to begin with the simplest solution and use workflows when paths are predefined, while reserving agents for situations requiring dynamic decisions about how to proceed. It balances consistency for high-volume normal cases with adaptable exception handling. See Anthropic's Building effective agents guidance and the Claude developer documentation.

QUESTION NO: 3

You are building a Claude application that needs to deliver model output to end users as it is generated, instead of waiting for the full response to complete.

The Claude API mechanism you would use is...

A. Structured JSON output, which delivers responses only after the model has finalized the JSON shape across the entire response.

B. Streaming responses, which deliver tokens incrementally as the model generates them so users see output progressively.

C. The Batch API, which delivers full responses after a delay suitable for non-interactive workloads.

D. Prompt caching, which speeds up the cost profile of future requests and does not affect the delivery timing of the first response.

ANSWER: B

Explanation:

Streaming responses, which deliver tokens incrementally as the model generates them so users see output progressively, is the appropriate mechanism for an interactive application that must display output before the entire model response is complete. With the Messages API, the application enables streaming by setting `stream: true`. Claude then sends a sequence of Server-Sent Events (SSE), allowing the client to process content blocks and text deltas as they become available rather than waiting for one completed response payload.

This reduces perceived latency: users can begin reading an answer almost immediately while Claude continues generating the remainder. A client can append each received text delta to a chat interface, track relevant lifecycle events, and use the final message-completion event to determine that generation has ended and obtain final metadata such as the stop reason and usage information. Streaming can also communicate incremental events for capabilities such as tool use, where applicable, so applications should consume the documented event stream rather than assume every event contains plain text.

Anthropic documents streaming as a first-class Messages API feature and provides streaming helpers in its official SDKs. See the Messages API streaming documentation and the Messages API reference.

QUESTION NO: 4

Your Claude agent performs database operations. A recent incident occurred where the agent ran a destructive query that affected production data. The team wants to add deterministic controls to prevent similar incidents.

How would you prevent similar incidents?

A. Run the agent only during business hours when humans are available to monitor its activity, treating the schedule as the primary control mechanism for destructive operations.

B. Add Claude hooks that intercept database operations and apply deterministic checks, such as blocking destructive queries or requiring approval, before the queries execute.

C. Switch to a higher-capability Claude model on the grounds that a more capable model is less likely to run destructive queries during normal operation across all requests.

D. Add a system prompt instruction telling the agent to be careful with database operations on every request the application handles during normal operation across all incoming traffic.

ANSWER: B

Explanation:

Add Claude hooks that intercept database operations and apply deterministic checks, such as blocking destructive queries or requiring approval, before the queries execute. This places an enforceable policy boundary between the agent's proposed action and the system that can cause an irreversible production side effect. The control does not depend on the model consistently interpreting a natural-language instruction correctly; instead, it evaluates concrete facts such as the target environment, database identity, query type, affected tables, and whether an approval token or authorized user is present.

In Claude Code, a `PreToolUse` hook runs before a tool call and can return a decision that allows, denies, or requests further handling of the proposed action. A database-operation hook can therefore reject statements matching prohibited patterns such as `DROP`, broad `DELETE`, or production schema changes; it can also route narrowly defined high-risk operations to an explicit approval workflow. Combined with a read-only default credential and least-privilege database roles, this makes destructive access unavailable unless the deterministic policy permits it.

This is the appropriate safety design because the execution boundary, rather than model behavior alone, is responsible for enforcing the rule. See Anthropic's Claude Code hooks documentation and its PreToolUse hook guidance.

QUESTION NO: 5

Your Claude application makes high-volume API calls during business hours and very few calls overnight. The team is concerned about staying within rate limits during peak hours and wants to understand how the Claude API enforces those limits.

How would you proceed?

A. Review the API documentation for streaming endpoints and evaluate whether migrating peak-hour calls to streaming reduces exposure to rate limit enforcement.

B. Assess the average payload size of current API calls and consolidate requests where possible to reduce the total number of calls made during peak hours.

C. Identify the rate limits, design the application to stay within them during peak hours, and use exponential backoff when limits are reached.

D. Examine the peak-hour request patterns in your application logs and smooth traffic by distributing requests more evenly across the business-hours window.

ANSWER: C

Explanation:

Identify the rate limits, design the application to stay within them during peak hours, and use exponential backoff when limits are reached. This is the appropriate approach because Claude API limits must be treated as concrete operating constraints during the application's busiest periods, rather than as an average calculated across an entire day. Anthropic applies limits across dimensions that include requests per minute, input tokens per minute, and output tokens per minute. The organization's available limits depend on its usage tier and model, so the team should first review the limits configured for the relevant workloads and estimate peak request and token demand against each applicable dimension.

The application should then implement admission control, queuing, throttling, or concurrency management so peak traffic remains within those boundaries. It should also be resilient when a limit is temporarily reached: HTTP 429 responses include retry guidance, and retrying with exponential backoff helps prevent immediate repeated requests from intensifying a short-

lived capacity constraint. Anthropic's SDKs also provide automatic retries for retryable failures by default, but teams should still design their workload and user-facing behavior around expected peak demand. See Anthropic's rate limits documentation and API error documentation.

QUESTION NO: 6

You are implementing a custom tool for your Claude agent. The tool needs to interact with an external pricing service that returns product data.

Which of the following best practices would you apply as you develop this tool?

A. Omit the tool description and let the model infer when to use the tool based on the tool's name and the rest of the prompt context.

B. Define the tool with a loose schema and let the model interpret the inputs flexibly on each call the agent makes.

C. Implement the tool with no error handling and let the agent loop catch failures whenever the pricing service returns an error during operation.

D. Define the tool with a clear schema, write a precise description for when to call it, and handle pricing service errors explicitly.

ANSWER: D

Explanation:

Define the tool with a clear schema, write a precise description for when to call it, and handle pricing service errors explicitly. This follows Anthropic's guidance for building dependable client-side tools. A tool definition should include a well-specified JSON Schema for its inputs, enabling Claude to form structured calls with unambiguous parameter names, types, and constraints. A detailed description is equally important because it tells Claude the tool's purpose, the situations in which it is appropriate to invoke it, how its parameters should be interpreted, and any relevant operational limitations. Clear tool descriptions improve selection accuracy when an agent has multiple possible actions.

Because the tool calls an external pricing system, the application should validate inputs and handle expected integration failures, such as timeouts, unavailable services, invalid responses, and rate limits. Returning a controlled error result to the agent allows it to respond appropriately, retry when suitable, request clarification, or offer a useful user-facing explanation. This creates a predictable boundary between Claude's reasoning and the external service while protecting the overall agent workflow from unhandled exceptions. Anthropic also documents strict tool use for cases where enforcing conformance to the declared input schema is especially important. See the Anthropic tool use documentation and the tool-use implementation guide.

QUESTION NO: 7

Your Claude application creates structured incident summaries from operational alerts. A deployment introduces an application bug that sends an unsupported request parameter, and the API consistently returns an invalid-request error. The retry wrapper currently retries every API failure three times before returning an error to the caller.

How would you improve the error-handling design?

A. Classify API failures by retryability, log the invalid request with relevant diagnostic context, and correct the request construction rather than retrying the same invalid call.

B. Increase the retry count so the application has more opportunities for the API to accept the unsupported parameter after a temporary service condition changes.

C. Retry invalid-request errors only after switching to a higher-capability Claude model, because model capability determines whether request parameters are accepted.

D. Suppress invalid-request errors in production logs because the caller already receives a failure after the configured retry attempts are exhausted.

ANSWER: A

Explanation:

Classify API failures by retryability, log the invalid request with relevant diagnostic context, and correct the request construction rather than retrying the same invalid call. An invalid-request response indicates that the application sent a request the API cannot accept. Repeating the unchanged request is unlikely to succeed and adds unnecessary latency, usage, and operational noise.

The integration should distinguish deterministic client-side failures from transient conditions. For example, malformed request parameters, authentication problems, and invalid input should follow a controlled non-retry path with useful diagnostics for developers. Transient failures such as selected server errors, network interruptions, or rate limits may justify bounded retries with appropriate backoff, subject to the applications requirements. Error records should include information such as the error type, status, request identifier where available, and a privacy-safe correlation identifier so the faulty deployment can be investigated without exposing sensitive alert content.

QUESTION NO: 8

You are building an agent that needs to call several internal APIs and a database in a structured, repeatable way. Your team has decided to use the Claude Agent SDK rather than build a custom loop. You are setting up the agent's tool definitions and execution loop.

How would you set up the tools and execution loop?

A. Use the SDK's tool interface and let the SDK handle the loop, dispatch, and history.

B. Call the Messages API directly and let the model format its tool calls in plain text.

C. Use the SDK's tool interface and loop, with conversation history stored in a separate team database.

D. Use the SDK's tool interface and write the loop and history layer in the team's own code.

ANSWER: A

Explanation:

Use the SDK's tool interface and let the SDK handle the loop, dispatch, and history. This approach matches the purpose of choosing the Claude Agent SDK: define the capabilities the agent may use as structured tools, then use the SDK's agent runtime to manage the repetitive agentic workflow. The SDK can expose tool schemas to Claude, process the model's tool-use requests, invoke the registered tool implementations, return tool results to the model, and continue execution until the task is complete. This provides a consistent execution model for calls to internal APIs and databases while keeping application code focused on the actual business operations performed by each tool.

For a repeatable workflow, tool definitions should have clear names, descriptions, and typed input parameters, with implementations that enforce authorization, validate inputs, and return useful structured results. The Agent SDK also provides first-class session support where persistent conversation state is needed, rather than requiring the application to recreate the basic turn-management machinery. This higher-level abstraction is especially appropriate when the goal is an agent that can coordinate multiple tool calls without implementing the low-level tool-use loop yourself. See Anthropic's Agent SDK overview and custom tools documentation.

QUESTION NO: 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:

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. This is the correct distinction because Agent Skills are intended to package reusable expertise and workflows for Claude. A Skill can contain instructions in its `SKILL.md` file along with supporting resources such as scripts, reference material, and templates. It is therefore a strong fit when a team wants to distribute a repeatable domain process—for example, document-generation conventions, data-analysis procedures, or a specialized business workflow—without requiring Claude to call a separately running external service.

An MCP server addresses a different integration boundary: it provides a standardized protocol through which a Claude client can discover and use externally provided capabilities. Those capabilities can include callable tools, contextual resources, and prompt templates, often backed by services, databases, APIs, or enterprise systems. In practice, the choice depends on whether the reusable asset is primarily packaged guidance and supporting files for Claude to apply, or an interface to live external capabilities and data. Teams may use both together: a Skill can provide the workflow guidance for deciding how to use tools supplied by an MCP server. Anthropic documents Skills as reusable, file-based capability packages and MCP as the protocol for connecting Claude to external tools and context.

See Anthropic Agent Skills overview and Anthropic MCP documentation.

QUESTION NO: 10

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:

Prompt injection, data leakage from prompts or context, jailbreak attempts, and unsafe model output that bypasses application controls are core AI-specific threat categories for a Claude application. Prompt injection occurs when adversarial instructions, whether supplied directly by a user or embedded in untrusted content such as retrieved documents, webpages, or tool results, attempt to override the application's intended behavior. Jailbreak attempts similarly seek to defeat model safeguards or application-level constraints. A threat model should also account for disclosure of sensitive information held in system prompts, conversation history, retrieved context, or connected-tool data. Finally, model output must be treated as potentially untrusted, especially when it is displayed to users, used in downstream workflows, or converted into tool calls or other actions. Anthropic recommends layered defenses: clear system instructions, input and output validation, isolating

untrusted content, minimizing tool permissions, requiring confirmation for consequential actions, and monitoring for suspicious behavior. These measures recognize that LLM applications introduce instruction-following and data-flow risks beyond the conventional web threats already included in the team's model. See Anthropic's jailbreak mitigation guidance and its system prompt guidance.

QUESTION NO: 11

Your Claude application requests structured JSON output from the model. Most of the time the JSON is well-formed, but occasionally Claude returns malformed JSON that breaks downstream processing.

How would you handle the malformed output?

A. Manually inspect every response before downstream processing so a human reviewer catches any malformed JSON before the application passes the response to downstream systems.

B. Add output validation that parses Claude's response against the expected schema and treats malformed output as a recognized error path with retry or fallback handling.

C. Switch to free-form text output so the application no longer depends on JSON parsing for any of the responses it sends to downstream systems during normal operation.

D. Retry the same request repeatedly until valid JSON appears in the model's response, with the retry loop adding delay to the application's response time on affected requests.

ANSWER: B

Explanation:

Add output validation that parses Claude's response against the expected schema and treats malformed output as a recognized error path with retry or fallback handling. Production applications should never assume that model-generated content is valid merely because a prompt requested a particular format. Instead, validate the response at the application boundary before sending it to downstream components. This means parsing the returned JSON and checking that it conforms to the fields, types, and constraints the application expects. If parsing or schema validation fails, the application should handle that condition deliberately—for example, by making a bounded retry, using a safe fallback, or returning a controlled error—rather than allowing malformed data to propagate and cause an uncontrolled failure. Anthropic recommends designing for failure modes and implementing robust error handling when building with Claude. Where appropriate, Claude's structured outputs capabilities can also help enforce a supplied JSON schema, but application-side validation and graceful handling remain essential safeguards for reliable systems. See Anthropic's output-formatting guidance and Messages API documentation.