CCDV-F · CHEAT SHEET · CERTIFYDECK

Claude Certified Developer – Foundations

Dominios, pesos oficiales y habilidades clave del temario de CCDV-F, para repasar antes del examen.

Nivel: Intermediate (Associate) Duración: 120 min Nota de corte: 720/1000 Fuente: guía de estudio oficial de Microsoft Learn

1. Agents and Workflows

17–17%

Architecture and Runtime Selection

- Usa workflows con rutas de código fijas cuando los pasos son previsibles y auditables, y agentes solo para problemas abiertos; añade complejidad solo si una evaluación lo justifica. Los subagentes en paralelo convienen en tareas amplias e independientes, no con contexto compartido. Agent SDK corre en tu proceso; Managed Agents aloja el bucle, en sandbox cloud o autoalojado.

Agent SDK Construction and Hooks

- En AgentDefinition, description decide la delegación y tools restringe herramientas; el subagente solo recibe el prompt de la herramienta Agent. Lo obligatorio va en hooks, no en el prompt: PreToolUse deniega antes de ejecutar, Stop con decision block impide terminar, y onFailure block evita que un fallo deje pasar la llamada. Los matchers comparan nombres de herramientas.

Agent Pattern Diagnosis

- Un agente es un modelo que usa herramientas en bucle: el bucle debe ramificar según stop_reason, tener una condición de parada y devolver los fallos con is_error. Los subagentes aíslan el contexto y solo reciben el prompt de delegación; la memoria externa, las listas de tareas y CLAUDE.md conservan lo que la compactación resumiría.

2. Applications and Integration

20–20%

Messages API Mechanics

- La Messages API no guarda estado: cada petición lleva el historial completo, los tool_result van primero en el mensaje de usuario y los bloques de thinking se devuelven sin cambios. El caché exige un prefijo idéntico (tools, system, messages) hasta el punto de ruptura, y Message Batches procesa de forma asíncrona y empareja resultados por custom_id.

Application Boundary and Surface Behavior

- Las aplicaciones de Claude añaden un system prompt propio que la API no aplica; en la API, el rol system tiene prioridad de operador sobre el de usuario y el contenido externo debe ir en tool_result. Claude solo recibe esquemas y resultados de tus herramientas; los archivos se aíslan por workspace y la PHI no va en esquemas JSON. Cada petición lleva su historial y las sesiones del SDK son locales salvo con sessionStore.

Configuration and SDLC Integration

- Los CLAUDE.md se concatenan (no se sustituyen) y los de subcarpetas cargan bajo demanda; en settings.json prima gestionado, línea de comandos, local, proyecto y usuario, y el model gestionado no bloquea: lo hace availableModels. Los ID de modelo son instantáneas fijas; en Managed Agents las sesiones se fijan a una versión del agente. En CI, --bare omite CLAUDE.md y hooks, y las cachés se aíslan por workspace.

Integration Resilience

- En streaming, stop_reason llega en message_delta, el usage es acumulativo, los deltas de herramienta son JSON parcial y los errores a mitad de stream llegan tras un 200 sin reintento del SDK. Se reintentan 429 con retry-after, 529 y 5xx según x-should-retry; un 400 o el tope de gasto no. Un rechazo es un 200 con stop_reason refusal, y la Files API exige el bloque adecuado a cada tipo de archivo.

3. Claude Code

8–8%

Claude Code Operation

- Los CLAUDE.md (gestionado, usuario, proyecto y CLAUDE.local.md) se concatenan, no se anulan, y son contexto, no imposición; para bloquear se usa permissions.deny. La prioridad es gestionada, --settings, local, proyecto y usuario. dontAsk con --allowedTools sirve para CI, --bare aísla claude -p y las sesiones en la nube solo ven lo subido al repositorio; Remote Control mantiene la ejecución en local.

Claude Code Customization and Extension

- CLAUDE.md y .claude/rules/ dan contexto permanente (las reglas con paths solo al tocar archivos que coinciden); las skills cargan su cuerpo bajo demanda y sustituyen a los comandos de barra; los subagentes aíslan el contexto y limitan herramientas; los hooks se ejecutan siempre en su evento. Un plugin empaqueta todo con .claude-plugin/plugin.json y se distribuye desde un marketplace.json.

4. Eval, Testing, and Debugging

10–10%

Evaluation Design

- Una buena evaluación parte de criterios de éxito específicos y medibles, usa calificadores de código cuando es posible y LLM-as-judge con rúbricas claras, calibrado frente a etiquetas humanas, cuando hace falta matiz. Las tareas deben ser inequívocas y tener solución de referencia, cubrir casos en que la conducta debe y no debe darse, y separar evaluaciones de capacidad y de regresión.

Failure-Mode Diagnosis

- Diagnosticar un fallo exige leer la traza: un rechazo llega como HTTP 200 con stop_reason refusal y se recupera en otro modelo o cambiando el prompt; las alucinaciones se reducen con citas y permiso para decir que no se sabe; muchos fallos nacen en la integración (historial no reenviado, texto tras tool_result, bloques de thinking modificados). La corrección debe atacar la causa, no reintentar el síntoma.

5. Model Selection and Optimization

12–12%

Generation Mechanics and Model Options

- Todo cuenta para la ventana de contexto (herramientas, resultados, caché) y los recuentos de tokens dependen del modelo. La salida no es determinista y la versión 2026-10-21 rechaza temperature, top_p y top_k. Con adaptive thinking, Claude decide en cada solicitud si piensa; effort es la palanca principal y flexible, mientras que max_tokens es el límite estricto que incluye el thinking.

Model Tier Selection and Release Planning

- Elige el nivel de modelo por coste por tarea completada en tu propio tráfico, no por precio por token, y ajusta el esfuerzo antes de cambiar de modelo. Empieza por el nivel más rápido si la latencia y el volumen mandan, o por el más capaz si manda la precisión. Los IDs de modelo son instantáneas fijas: cambia solo tras aprobar tu conjunto de regresión y una prueba en sombra.

Cost Diagnosis and Optimization

- Lee los campos de uso de la respuesta, incluidas las lecturas y escrituras de caché, para ver qué domina el coste. La caché es una coincidencia exacta del prefijo tools, system y messages: pon el punto de ruptura en el último bloque estable y añade otro si los turnos crecen tanto que la búsqueda hacia atrás no llega a la escritura previa. Lo que no es urgente va a Message Batches, y la salida se reduce pidiendo respuestas más cortas.

6. Prompt and Context Engineering

15–15%

Context Management for Long-Running Sessions

- Al crecer el contexto, la precisión y el recuerdo empeoran (context rot), así que hay que curarlo. La compactación resume los turnos antiguos, la limpieza de resultados retira salidas viejas de herramientas, los subagentes aíslan la salida extensa y devuelven un resumen, y las notas persistentes dan continuidad entre sesiones. En Claude Code, /clear entre tareas no relacionadas o tras dos correcciones fallidas.

Prompt Revision

- Antes de revisar un prompt hacen falta criterios medibles y una suite automática que se vuelva a ejecutar entera tras cada cambio. Las correcciones documentadas son pedir la acción y el formato en positivo y con su motivo, usar ejemplos variados en etiquetas <example>, poner los documentos largos antes de la pregunta, dejar el contenido no fiable en tool_result codificado en JSON y sustituir el prefill por salidas estructuradas.

Structured Output and Defensive Parsing

- output_config.format garantiza JSON válido según el esquema y strict: true garantiza las entradas de herramienta, pero el código debe seguir comprobando stop_reason (refusal, max_tokens), comparar enums sin distinguir mayúsculas y validar las restricciones que el SDK traslada a las descripciones. Con citas no se puede usar output_config.format: se piden líneas "campo: valor" y se analizan de forma defensiva.

Grounding and Source Citation

- Para anclar respuestas se activan las citas en los documentos o en bloques search_result (referencias válidas y cited_text sin coste de salida), se elige el tipo de documento según la granularidad y se da permiso para responder "no encontrado" limitando al modelo a los documentos. Si el corpus cabe y es estable, se envía entero con caché de prompts; si es grande o cambia a menudo, se recuperan solo los pasajes relevantes.

7. Security and Safety

8–8%

Prompt Injection and Jailbreak Defense

- La inyección directa viene del propio usuario; la indirecta llega en contenido de terceros (correos, webs, OCR, resultados de herramientas). Contra la indirecta: entregar ese contenido solo en bloques tool_result, indicar su origen, declarar en el system prompt que es dato no fiable, codificarlo en JSON, filtrarlo con un clasificador ligero y aplicar privilegio mínimo; después, red teaming y monitorización.

Deterministic Guardrails and Least Privilege

- Una instrucción en el prompt no impide una acción destructiva; lo hacen las reglas deny, los hooks PreToolUse que bloquean (exit 2 o permissionDecision deny, con onFailure: "block" si el guardián puede fallar) y el sandbox, que limita a nivel de sistema operativo lo que el comando ya no puede evitar. Las reglas deny se aplican en todos los modos, incluso bypassPermissions, y las de la configuración gestionada no se pueden anular.

API Key and Credential Handling

- La clave vive en el back end, nunca en el navegador, y en agentes conviene que un proxy externo la inyecte para que el agente no la vea. Usa claves personales para tu trabajo, de cuenta de servicio para cargas compartidas y federación de identidad (WIF) donde haya un proveedor de identidad; separa entornos por espacio de trabajo, guarda las claves en un gestor de secretos, revisa su uso y caducidad y revoca de inmediato una clave filtrada.

8. Tools and MCP

10–10%

Tool Design and Response Shaping

- La descripción es lo que más influye en una herramienta: qué hace, cuándo usarla y cuándo no, sus parámetros y qué no devuelve. Las respuestas deben ser breves (campos útiles, identificadores legibles, paginación y truncado con indicaciones) y los errores, accionables y con is_error. Las herramientas propias se ejecutan en tu código y las de servidor en Anthropic; con muchas, ayudan los prefijos por servicio y la búsqueda de herramientas.

MCP Server Development and Integration

- Un servidor MCP ofrece herramientas (las invoca el modelo), recursos (datos de solo lectura que aporta la aplicación) y prompts (plantillas que elige el usuario). Stdio es un subproceso local que solo escribe mensajes JSON-RPC en stdout y toma las credenciales del entorno; Streamable HTTP sirve a muchos clientes remotos con OAuth, validando que cada token se emitió para ese servidor.

Capability Mechanism Selection

- Las herramientas de servidor (web_search, code_execution) las ejecuta Anthropic; las de esquema de Anthropic (bash) y las propias las ejecuta tu aplicación, ideales para lógica privada. Una Skill aporta procedimientos que Claude carga bajo demanda según su descripción, y un servidor MCP conecta un servicio externo que su equipo mantiene y comparten varios clientes.