CCAR-F · CHEAT SHEET · CERTIFYDECK

Claude Certified Architect – Foundations

Dominios, casos de estudio y guía rápida de repaso de cara al examen.

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

1. Agentic Architecture & Orchestration 27–27%

Design and implement agentic loops for autonomous task execution

- En un bucle propio sobre la Messages API, stop_reason decide el flujo: mientras valga tool_use se ejecutan las herramientas, se devuelven los tool_result y se vuelve a llamar a la API; con end_turn la respuesta es final. Otros valores necesitan su propio tratamiento: max_tokens indica truncado y pause_turn obliga a reenviar la conversación.
- Entre iteraciones se añade al historial la respuesta del asistente sin cambios y después un mensaje de usuario con un bloque tool_result por cada tool_use, enlazado por tool_use_id, también cuando la herramienta falla (con is_error). Los tool_result van juntos en un único mensaje y antes de cualquier texto. Así Claude incorpora la información nueva y decide su siguiente paso.
- En el Claude Agent SDK el bucle es automático y el final se lee en el subtype del ResultMessage.
- La finalización del bucle debe basarse en stop_reason, no en el texto del modelo. Buscar frases como tarea completada, tratar cualquier bloque de texto como respuesta final o usar un tope fijo de iteraciones como mecanismo principal provoca cortes prematuros o respuestas incompletas, porque una respuesta tool_use puede traer texto.

Orchestrate multi-agent systems with coordinator-subagent patterns

- Un coordinador eficaz analiza cada consulta y decide qué subagentes necesita, en lugar de pasar siempre por toda la cadena. Anthropic recomienda ajustar el esfuerzo a la complejidad (un agente para un dato sencillo, varios para comparaciones amplias) y, en el Agent SDK, Claude elige subagente según su campo description. Límites como maxBudgetUsd acotan el gasto de forma determinista.
- Para evitar duplicados y huecos, el coordinador reparte el alcance en partes sin solapamiento (por subtema, periodo, región o tipo de fuente) y da a cada subagente un objetivo, un formato de salida, orientación sobre herramientas y fuentes y límites claros. Anthropic observó que las instrucciones vagas hacían que varios subagentes repitieran las mismas búsquedas. En el Agent SDK, tools y mcpServers acotan las fuentes de cada subagente.
- En un bucle de refinamiento, el coordinador evalúa la síntesis frente a lo que pidió el usuario, convierte cada hueco en consultas concretas para los subagentes de búsqueda y análisis y vuelve a invocar la síntesis hasta que la cobertura es suficiente. Funciona cuando hay criterios de evaluación claros y la retroalimentación mejora el resultado; un tope de rondas solo hace de salvaguarda.
- Los topes de turnos o de gasto sirven solo como red de seguridad.

Configure subagent invocation, context passing, and spawning

- Hacer pasar toda la comunicación por el coordinador le da un único punto para observar cada traspaso, aplicar la misma política ante errores o resultados parciales y decidir qué información recibe cada subagente. En el Agent SDK, un subagente solo devuelve su mensaje final al padre y solo recibe el prompt de la herramienta Agent; los hooks SubagentStart y SubagentStop permiten auditar cada delegación.
- Un subagente que no es una bifurcación empieza con un contexto nuevo: no ve la conversación ni los resultados de herramientas del coordinador y solo recibe el prompt de la herramienta Agent. Por eso el coordinador debe incluir en ese prompt los hallazgos completos de los agentes anteriores (resultados de búsqueda, análisis, rutas de archivo, errores y decisiones), sin resúmenes que pierdan datos.
- Al pasar contexto entre agentes conviene separar el contenido de sus metadatos (URL, nombre de documento, página) con un formato estructurado: etiquetas XML como document, source y document_content, registros con campos propios o bloques search_result con source y title, que permiten citas automáticas. Así cada agente conserva la atribución sin tener que deducirla del texto.
- Para ejecutar subagentes en paralelo, el coordinador emite varias llamadas a la herramienta Agent (que antes se llamaba Task) en una misma respuesta, de modo que el tiempo total es el del subagente más lento y no la suma.
- El coordinador debe indicar a los subagentes el objetivo, su contexto y los criterios de calidad (qué fuentes valen, qué debe contener una buena respuesta) en lugar de un procedimiento paso a paso. Anthropic comprobó que en investigación, donde el camino no se puede prever, las heurísticas funcionan mejor que las reglas rígidas, y que unos criterios de éxito claros ayudan a saber cuándo parar.
- Solo conviene con subtareas independientes y autocontenidas; si una etapa depende de otra, como la síntesis de las búsquedas, se lanza cuando llegan sus resultados.

Implement multi-step workflows with enforcement and handoff patterns

- Cuando un paso debe ocurrir siempre antes que otro, la garantía no puede depender del prompt: un hook PreToolUse sobre la herramienta posterior puede devolver permissionDecision "deny" mientras falte el requisito, y Claude lee el permissionDecisionReason para corregir el rumbo. Un hook PostToolUse puede leer tool_response de la herramienta previa y registrar el dato verificado por session_id.
- Una petición con varios problemas se divide en elementos independientes que se investigan en paralelo y se resuelven en una sola respuesta. Claude puede emitir varios bloques tool_use en un turno; todos los tool_result vuelven juntos en un único mensaje de usuario. Un subagente empieza sin el historial del padre: el prompt de la herramienta Agent es lo único que recibe, así que debe llevar los datos ya verificados.
- Si la persona que recibe el escalado no ve la conversación, el traspaso debe ser autosuficiente: una herramienta de escalado cuyo input_schema exija customer_id, causa_raíz, importe y acción recomendada. Con strict: true la entrada siempre cumple el esquema (campos obligatorios y tipos) y unas descripciones detalladas de cada parámetro orientan el contenido; un hook PreToolUse puede comprobar las cifras contra el sistema de registro.

Apply Agent SDK hooks for tool call interception and data normalization

- Un hook PostToolUse recibe tool_input y tool_response y puede devolver updatedToolOutput para sustituir el resultado antes de que Claude lo vea; así se unifican fechas Unix e ISO 8601 o códigos de estado numéricos de distintos servidores MCP. decision "block" y additionalContext solo añaden texto junto al resultado original. Las herramientas MCP se filtran con mcp__<servidor>__*.
- Un hook PreToolUse lee tool_input antes de que la herramienta se ejecute y puede devolver "deny" con un permissionDecisionReason que Claude lee y usa para tomar otra vía, como escalar a una persona. Si varios hooks deciden distinto, gana el más restrictivo: deny, luego defer, ask y allow. Con "defer" el turno termina con stop_reason "tool_deferred" y la llamada pendiente puede reanudarse tras una aprobación externa.
- Prompt, CLAUDE.md y skills son instrucciones no deterministas: Claude suele seguirlas, pero sin garantía. Los hooks, las reglas deny y la configuración gestionada los aplica el entorno siempre. Para una regla sin excepciones se usa un hook command, http o mcp_tool (un hook prompt o agent deja decidir a otro modelo) y onFailure "block" si el hook puede fallar; las preferencias se quedan en CLAUDE.md.

Design task decomposition strategies for complex workflows

- El encadenamiento de prompts divide una tarea en pasos fijos: cada llamada procesa la salida de la anterior y se pueden añadir comprobaciones programáticas entre pasos; cambia latencia por precisión. Cuando no se pueden prever las subtareas, el patrón orquestador-trabajadores deja que un modelo central las decida según la entrada, las delegue y sintetice los resultados. Lo recomendable es la solución más simple que funcione.
- El contexto es un recurso finito: cuantos más tokens, menos precisión al recordar y relacionar información. En revisiones grandes conviene una pasada local por archivo, centrada en un solo aspecto, y una pasada de integración separada que reciba resúmenes condensados (problemas e interfaces cambiadas) para detectar fallos entre archivos. Cada subagente recibe solo el prompt de la herramienta Agent y devuelve solo su mensaje final.
- Una tarea abierta como añadir pruebas a un código heredado se aborda explorando primero la estructura, identificando las zonas de más impacto y redactando un plan priorizado que se ajusta con lo que se descubre. La exploración puede delegarse a un subagente y el plan revisarse en plan mode antes de editar. En trabajos de varias sesiones ayuda un archivo de tareas con estado y notas de progreso.

Manage session state, resumption, and forking

- Claude Code guarda cada conversación: claude --continue reabre la más reciente del directorio, mientras que nombrar la sesión con claude -n o /rename permite volver a ella con claude --resume <nombre> o /resume <nombre>. Un nombre ambiguo abre el selector. Al reanudar no se restauran flags como --mcp-config, --add-dir o --allowed-tools, que hay que volver a pasar; el nombre por defecto de la sesión no sirve para reanudar.
- En el Agent SDK, resume con fork_session=True (forkSession en TypeScript) crea una sesión nueva que parte de una copia del historial original: la bifurcación recibe su propio session_id y la original no cambia. Bifurcar copia la conversación, no los archivos: las ediciones de una rama son reales en el directorio. En la CLI se bifurca con /branch o con --resume/--continue más --fork-session.
- Reanudar una sesión (resume por ID, continue para la más reciente) devuelve al agente todo su historial, incluidos los resultados de herramientas: conviene cuando ese contexto sigue siendo válido, por ejemplo tras un límite de turnos. Si los datos consultados han cambiado, es mejor iniciar una sesión nueva con un resumen de las conclusiones. Fork copia el historial para probar otra vía sin tocar el original.
- Una sesión guarda la conversación, no el sistema de archivos: al reanudarla, el agente conserva las lecturas antiguas de los archivos como resultados pasados y no sabe que han cambiado. Para un reanálisis dirigido, el prompt de reanudación debe nombrar los archivos modificados y pedir que se relean y se revisen solo las conclusiones afectadas, en lugar de volver a explorar todo o rebobinar con file checkpointing.

2. Tool Design & MCP Integration

18–18%

Design effective tool interfaces with clear descriptions and boundaries

- La descripción es el factor que más influye en cómo usa Claude una herramienta: debe explicar qué hace, cuándo usarla y cuándo no frente a herramientas parecidas, qué significa cada parámetro y qué no devuelve. Los parámetros deben tener nombres inequívocos (user_id mejor que user), input_examples ayudan con entradas complejas, pero no sustituyen una descripción clara.
- Cuando dos herramientas se solapan o tienen un propósito vago, el agente duda o elige mal. La solución es darles nombres que reflejen su función real, con prefijos por servicio o recurso cuando proceda (tickets_search, wiki_search) y descripciones que delimiten su alcance y remitan a la alternativa. Después hay que medir el cambio con una evaluación y revisando transcripciones.
- Una herramienta genérica cuyo comportamiento y salida dependen de un parámetro de texto obliga al agente a adivinar qué recibirá. Es mejor dividirla en herramientas con un propósito claro y contratos propios de entrada y salida (en MCP, outputSchema con structuredContent), como extraer datos, resumir o verificar una afirmación contra una fuente. Las herramientas que solo envuelven endpoints de una API son un error habitual.
- El system prompt y las definiciones de herramientas comparten el contexto, así que una regla basada en una palabra clave ("si el usuario dice X, llama a Y") puede anular una descripción bien escrita y provocar llamadas indebidas. Al revisar un system prompt hay que buscar esas reglas frágiles y sustituirlas por criterios de intención coherentes con las descripciones, en lugar de añadir énfasis o clasificadores.

Implement structured error responses for MCP tools

- En MCP, los fallos de ejecución (API caída, entrada no válida, regla de negocio) se devuelven en el resultado con isError: true para que el modelo pueda corregirse; los errores de protocolo JSON-RPC quedan para fallos de la propia petición. El contenido del error debe decir la categoría (transitorio, validación o permiso), si reintentar tiene sentido y qué ha fallado, en lugar de un 'Error' genérico.
- Una infracción de una regla de negocio (fuera de plazo, póliza no vigente) fallará siempre, así que el error debe marcarse como no reintentable y llevar una explicación comprensible que el agente pueda transmitir al cliente. Lo que devuelve la herramienta entra en el contexto, de modo que los detalles internos deben quedarse en los registros del servidor y la salida debe sanearse.
- Los fallos transitorios deben resolverse dentro del subagente con reintentos limitados; al coordinador solo debe llegar lo que no se puede resolver localmente. Como solo el mensaje final del subagente vuelve al padre, ese informe debe incluir los resultados parciales, el error pendiente y lo que se intentó. Tras un isError, Claude puede reintentar, así que el bucle debe limitar los reintentos.
- Un fallo de acceso (permiso denegado, tiempo agotado) y una consulta correcta sin coincidencias significan cosas distintas y deben notificarse de forma distinta. El primero va con isError: true y su categoría, para que el agente decida si reintenta o escala; el segundo es un resultado correcto que dice que no hay datos. Mezclarlos lleva a respuestas falsas o a reintentos y escalados innecesarios.

Distribute tools appropriately across agents and configure tool choice

- Cada subagente debe tener solo las herramientas de su rol: en el Claude Agent SDK se fijan con el campo tools de AgentDefinition (o se quitan con disallowedTools); si se omite tools, el subagente hereda todas las disponibles. allowedTools solo aprueba llamadas automáticamente y no oculta las demás herramientas, y una instrucción en el prompt no impide que el subagente use una herramienta que tiene.
- Una herramienta genérica (fetch_url, run_sql) da al agente más capacidad de la que necesita su rol. Sustituirla por una herramienta restringida, como un load_document que solo acepte URL del repositorio, permite al servidor validar cada entrada y rechazar lo que no corresponde, y devolver solo información relevante. No impide que el modelo lo intente, pero sí que la petición prospere.
- Separar herramientas por rol no obliga a pasar todo por el coordinador. Para necesidades frecuentes y simples de otro rol (comprobar un dato, leer un plan) conviene dar una herramienta acotada, a menudo de solo lectura y con un contrato estricto. Las cosas complejas o sensibles, como investigar temas nuevos o modificar datos, siguen pasando por el coordinador, que es quien planifica y aprueba.
- Con tool_choice en Claude, el modelo debe llamar a una de las herramientas proporcionadas, aunque elige cuál; tool fuerza una herramienta concreta, auto deja decidir y none impide usarlas. Con any o tool la respuesta empieza por la llamada, sin texto previo. Combinado con strict: true garantiza además entradas válidas, y no todos los modelos admiten el uso forzado.

Integrate MCP servers into Claude Code and agent workflows

- Los servidores MCP de ámbito de proyecto se definen en .mcp.json en la raíz del repositorio, bajo mcpServers, y se suben a git para compartirlos. Admiten ${VAR} y ${VAR:-default} en command, args, env, url y headers, de modo que cada persona aporta su token. Claude Code pide aprobación en sesiones interactivas; una variable vacía cuenta como definida.
- Los servidores MCP personales o experimentales se guardan en ~/.claude.json. El ámbito local, que es el predeterminado de claude mcp add, los carga solo en el proyecto actual; --scope user los ofrece en todos tus proyectos, sin compartirlos. Si un nombre se repite, gana local, luego proyecto y luego usuario, y la entrada ganadora se usa entera, sin mezclar campos.
- Claude elige herramientas sobre todo por su descripción: si una herramienta MCP solo dice "busca símbolos", Claude seguirá usando Grep. Describe qué abarca, qué devuelve y cuándo usarla, con las palabras clave al principio, porque la búsqueda de herramientas comparte con la primera parte. Las instrucciones del servidor indican a Claude cuándo buscar sus herramientas.
- Para integraciones estándar, como un gestor de incidencias o el alojamiento de código, conviene añadir el servidor MCP que ya existe con claude mcp add (Anthropic Directory, MCP Registry) en lugar de escribir uno. Reserva los servidores propios para flujos del equipo que nadie cubre, con pocas herramientas que resuelvan tareas completas y no simples envoltorios de endpoints.
- Los recursos MCP son datos de solo lectura con URI propia que la aplicación lista con resources/list o descubre como plantillas parametrizadas con resources/templates/list. Exponer un catálogo así evita llamadas exploratorias. En Claude Code se citan con @servidor:protocolo://ruta y se adjuntan, y Claude recibe herramientas para listarlos y leerlos.

Select and apply built-in tools (Read, Write, Edit, Bash, Grep, Glob) effectively

- Grep busca patrones dentro de los archivos con la sintaxis regex de ripgrep: sirve para encontrar llamadas a una función o el origen de un mensaje de error. Sus modos son files_with_matches (por defecto), content y count, y admite filtros glob y type. Respeta .gitignore, así que un archivo ignorado se busca pasando su ruta.
- Glob busca archivos por patrón de nombre, no por contenido: **/*.test.tsx encuentra pruebas a cualquier profundidad y *.{json,yaml} solo en el directorio actual. Los resultados se ordenan por fecha de modificación y tienen un límite con aviso de truncado, así que conviene acotar el patrón. A diferencia de Grep, no respeta .gitignore por defecto.
- Edit sustituye un old_string que debe aparecer exactamente una vez: si se repite, se amplía el contexto o se usa replace_all. Cuando no existe un ancla única, se lee el archivo completo con Read y se guarda el contenido entero con Write, que sobrescribe y nunca combina. Una lectura parcial con aviso PARTIAL view no basta: hay que leer el resto antes.
- Claude Code carga CLAUDE.md al principio y explora el resto justo a tiempo: localiza un punto de entrada con Grep, por ejemplo una ruta o un texto de error, y después sigue los imports con Read abriendo solo los archivos del camino. Leerlo todo de entrada llena el contexto de código irrelevante y empeora las respuestas, aunque la ventana sea grande.
- Cuando un módulo envuelve o reexporta funciones, a veces con alias desde un archivo barrel, buscar el nombre original o la ruta de import se queda corto. Primero se lee con Read el envoltorio y el barrel para listar todos los nombres exportados y sus alias, y después se busca cada nombre con Grep en todo el código para encontrar las llamadas reales.

3. Claude Code Configuration & Workflows

20–20%

Configure CLAUDE.md files with appropriate hierarchy, scoping, and modular organization

- Las instrucciones de CLAUDE.md tienen ámbitos: política gestionada, usuario (~/.claude/CLAUDE.md, solo para ti), proyecto (./CLAUDE.md o ./.claude/CLAUDE.md, compartido por git) y local (CLAUDE.local.md, personal). Si un compañero no recibe una regla, suele estar en un archivo de usuario. Los CLAUDE.md de subdirectorios se cargan bajo demanda.
- Un CLAUDE.md puede importar otros archivos con @ruta; las rutas relativas se resuelven desde el archivo que importa, no desde el directorio de trabajo. Así cada responsable de paquete incluye solo los estándares que su código necesita desde una fuente común. Ojo: lo importado se carga con el CLAUDE.md y no ahorra contexto; para eso están las reglas con paths.
- Un CLAUDE.md grande se divide en archivos de un solo tema en .claude/rules/, como testing.md o deployment.md, que se descubren de forma recursiva. Sin frontmatter se cargan al arrancar; con paths solo cuando Claude lee o edita archivos que coinciden, y paths es el único campo que se lee. Siguen siendo contexto, no configuración obligatoria.
- /memory lista las ubicaciones de CLAUDE.md y CLAUDE.local.md de usuario y proyecto, aunque aún no existan, y abre cada archivo en el editor; también da acceso a la carpeta de auto memory y a su interruptor. Para confirmar qué se cargó en la sesión se usa /context all (Memory files). Las diferencias entre personas suelen venir de archivos de usuario o de la auto memory.

Create and configure custom slash commands and skills

- Un archivo Markdown en .claude/commands/ del proyecto define un comando que se comparte con el equipo a través del control de versiones: deploy.md se invoca como /deploy y una subcarpeta añade un prefijo con dos puntos (/frontend:component). Los comandos se han fusionado con las skills, para añadir scripts se convierte el comando en .claude/skills/<nombre>/SKILL.md, y ~/.claude/commands/ queda para uso personal.
- Con context: fork en el frontmatter, la skill se ejecuta en un subagente nuevo (el tipo lo fija agent, por defecto general-purpose) que recibe el contenido de SKILL.md como tarea, no ve el historial de la conversación y devuelve un resumen. Sirve para aislar salidas extensas o exploratorias, se ejecuta en segundo plano salvo background: false, y Explore arranca sin CLAUDE.md.
- En Claude Code, allowed-tools del frontmatter de una skill preaprueba las herramientas indicadas solo durante el turno que la invoca, sin limitar las demás: siguen disponibles bajo la configuración de permisos. Para restringir herramientas mientras la skill está activa se usa disallowed-tools, y para bloquearlas en todas partes, reglas de denegación, una skill de proyecto puede autoconcederse acceso amplio, así que conviene revisarla.
- El campo argument-hint del frontmatter muestra durante el autocompletado los argumentos que espera una skill, por ejemplo [issue-number], tras el nombre del comando. Es solo una pista: no valida la entrada, los marcadores indexados sin argumento quedan literales y los declarados en arguments se vacían. Es una extensión de Claude Code que las subidas a claude.ai y la Skills API rechazan.
- CLAUDE.md se carga en cada sesión y sirve para hechos y reglas que Claude debe tener siempre presentes: comandos de compilación, convenciones y prohibiciones. Las skills se cargan bajo demanda y son para material de referencia o flujos que se lanzan con /nombre: un patrón habitual combina una línea en CLAUDE.md con la guía completa en una skill. Las importaciones @path no reducen el contexto.

Apply path-specific rules for conditional convention loading

- Un archivo Markdown en .claude/rules/ con frontmatter YAML paths (por ejemplo paths: ["terraform/**/*"]) solo se carga cuando Claude lee, escribe o edita un archivo que coincide, también con cat o head en Bash. paths es el único campo que Claude Code lee de una regla; sin él, la regla se carga al iniciar como .claude/CLAUDE.md. Las reglas son orientación, no configuración obligatoria.
- El campo paths de una regla admite patrones glob relativos al directorio que contiene la carpeta .claude: **/*.test.tsx alcanza todos los archivos de prueba en cualquier carpeta, mientras que src/components/*.tsx solo coincide con archivos directos de esa carpeta y *.md solo con los de la base. Se pueden listar varios patrones y usar llaves, como src/**/*.{ts,tsx}.
- Tanto un CLAUDE.md por directorio como una regla con paths en .claude/rules/ dirigen instrucciones a una parte del árbol. El CLAUDE.md de subdirectorio vive junto al código, lo mantienen sus responsables y se carga al tocar cualquier archivo de esa carpeta; la regla vive en el .claude/ central y conviene cuando la misma convención se aplica a muchas rutas dispersas o a un tipo de archivo.

Determine when to use plan mode vs direct execution

- Plan mode hace que Claude lea archivos, ejecute comandos de exploración y proponga un plan sin editar el código hasta que lo apruebas. Conviene en tareas con implicaciones de arquitectura, como migraciones que tocan decenas de archivos o elegir entre enfoques de integración, porque evita rehacer trabajo si la dirección inicial es mala. Se activa con Shift+Tab, /plan, --permission-mode plan o defaultMode en .claude/settings.json.
- Explore es un subagente integrado de solo lectura (Write y Edit denegados) para descubrir archivos y buscar código: trabaja en su propia ventana de contexto y devuelve solo un resumen, lo que evita agotar el contexto principal en tareas de varias fases. Arranca sin CLAUDE.md ni estado de git, así que las reglas imprescindibles deben repetirse al delegar, para explorar y modificar se usa general-purpose.
- Una pauta habitual es investigar en plan mode y ejecutar después: Claude explora y propone un plan, puedes pedir cambios con No, keep planning o editarlo con Ctrl+G, y al aprobarlo la sesión sale de plan mode y Claude empieza a editar. Con showClearContextOnPlanAccept se puede aprobar y limpiar el contexto de la planificación, y el plan se reinyecta desde disco tras compactar.
- Cuando una descripción en lenguaje natural da resultados incoherentes, unos pocos ejemplos concretos de entrada y salida (few-shot) suelen ser la forma más fiable de fijar el formato y la transformación esperada. Deben ser relevantes, lo bastante diversos para cubrir casos límite sin enseñar patrones no deseados, e ir en etiquetas example dentro de un bloque examples para separarlos de las instrucciones.

Apply iterative refinement techniques for progressive improvement

- Escribir antes las pruebas del comportamiento esperado, los casos límite y los requisitos de rendimiento da a Claude una definición de terminado con una comprobación de aprobado o fallido puede cambiar el código, ejecutarla, leer los fallos y corregirlos. Conviene poner el comando de pruebas en CLAUDE.md y pedir el comando y su salida como prueba, un hook Stop con código 2 hace que el entorno lo imponga.
- En dominios poco conocidos conviene pedir a Claude que te entreviste antes de implementar: plantea preguntas sobre lo que no has decidido, como estrategias de invalidación de caché o modos de fallo, hasta poder escribir una especificación en SPEC.md. Después se abre una sesión nueva que construye a partir de ese archivo, las tareas programadas no pueden hacer preguntas, así que la entrevista va antes.
- Cuando un caso límite falla, como los nulos de un script de migración, describe la salida que esperas con entradas concretas y conviértelas en tests: así Claude tiene una comprobación de aprobado o fallo con la que iterar. Si tras dos correcciones sigue mal, usa /clear y un único prompt nuevo con lo aprendido, y aclara que los tests verifican la regla, no la sustituyen, para evitar valores codificados a mano.
- Si dos arreglos interactúan, descríbelos juntos en un único mensaje detallado que explique su relación y el resultado combinado esperado, con un escenario que sirva de test. Los problemas independientes se resuelven mejor de uno en uno, cada uno con su comprobación y con /clear entre tareas no relacionadas. Tras dos correcciones fallidas, empieza de nuevo con un prompt que recoja lo aprendido.

Integrate Claude Code into CI/CD pipelines

- En CI, ejecuta Claude Code con -p (--print) procesa el prompt sin sesión interactiva, imprime el resultado y termina con un código de salida que el script puede comprobar. Las llamadas que necesitarían aprobación se deniegan salvo que algo las apruebe, así que preapruébalo lo necesario con --allowed-tools o un modo de permisos, y usa --permission-prompts-none cuando nadie puede responder a un host de permisos.
- Para publicar hallazgos como comentarios en línea, ejecuta claude -p con --output-format json y --json-schema: la respuesta incluye los metadatos de la sesión y los datos conformes al esquema en el campo structured_output. Si la ejecución termina sin salida válida, ese campo no aparece, así que compruébalo antes de publicar; un esquema no válido provoca un error y la palabra clave format no se aplica.
- Claude solo puede evitar escenarios de test duplicados si ve los tests existentes: pásale los ficheros que ejercitan el módulo modificado o pídele que los lea, y pide solo escenarios que no estén cubiertos. Un contexto centrado en esos ficheros funciona mejor que la suite entera, recuerda que el modo bare no carga CLAUDE.md, así que las instrucciones deben ir en el prompt.
- Documenta en CLAUDE.md lo que Claude no puede deducir del código: qué tests aportan valor, qué omitir y qué fixtures usar, con instrucciones concretas y un fichero breve. Si la guía solo importa en tests/, ponla en .claude/rules/ con paths para que se cargue al trabajar con esos ficheros. En CI con --bare, CLAUDE.md no se carga, así que pasa los estándares con --append-system-prompt-file.

4. Prompt Engineering & Structured Output

20–20%

Design prompts with explicit criteria to improve precision and reduce false positives

- Para reducir falsos positivos, define de forma explícita qué notificar (errores lógicos, seguridad, fugas de datos) y qué omitir (estilo, lo que el CI ya aplica, ficheros generados), y exige una cita fichero:línea para las afirmaciones de comportamiento. Si una categoría genera mucho ruido, exíla temporalmente en REVIEW.md mientras mejoras sus criterios, sin apagar el resto de la revisión.
- Para que la gravedad se clasifique de forma coherente, define cada nivel con ejemplos de código concretos, incluidos los casos dudosos y el motivo de su nivel, porque los ejemplos funcionan mejor que las instrucciones abstractas. Un enum en el esquema solo garantiza una etiqueta válida, no la correcta. En Code Review, la calibración se ajusta en REVIEW.md, indicando qué es importante y qué es ruido en tu caso.

Apply few-shot prompting to improve output consistency and quality

- Para los casos ambiguos, añade unos pocos ejemplos dirigidos que muestren la situación, la acción elegida y por qué se descartaron las alternativas plausibles, ya que Claude generaliza a partir de la explicación. Los ejemplos deben parecerse al caso real: variar lo suficiente para no transmitir patrones superficiales en etiquetas example dentro de examples, separados de las instrucciones.
- Unos pocos hallazgos de ejemplo en el formato exacto (ubicación como fichero:línea, problema, gravedad y corrección concreta) consiguen más consistencia que describir el formato en abstracto. Si un parser necesita que los campos estén siempre presentes, usa salidas estructuradas o strict: true que garanticen el esquema; los ejemplos siguen siendo necesarios para que el contenido de cada campo tenga el estilo deseado.
- Para reducir falsos positivos sin perder errores reales, usa ejemplos contrastados: un patrón aceptable y otro señalado que parecen casi idénticos, con la razón de cada decisión. Varía servicios y formas de código para que Claude no se ajuste a patrones superficiales, y cubre los casos límite explícitamente.
- Si los documentos varían en estructura, pero citas, en línea frente a bibliografía, o metadatos en su propia sección frente a datos dentro del texto, incluye ejemplos de cada estructura con la extracción correcta. En documentos largos, colócalos al principio del prompt, por encima de instrucciones y ejemplos, envueltos en etiquetas y pide a Claude que cite los pasajes relevantes antes de extraer.
- Ordena un campo "relevancia" o "limitaciones" en el documento, el problema suele ser un lío que Claude no ha visto: añade ejemplos de esos formatos con el valor correcto y describe en el esquema qué aspecto tiene el campo. Incluye también un ejemplo en el que el dato falta de verdad y null es correcto. strict: true solo garantiza la forma, no que valor se extrae.

Enforce structured output using tool use and JSON schemas

- Para extraer datos con tool use, define una herramienta cuyo input_schema describa los campos y lee el objeto input del bloque tool_use de la respuesta (stop_reason tool_use), localizándolo por type y name, porque puede ir precedido de un bloque de texto. Con strict: true el modelo se restringe para que las entradas cumplan el esquema, sin bucles de validación y reintento.
- Para datos de extracción y un tipo de documento desconocido, tool_choice {type: any} obliga a Claude a llamar a alguna de las herramientas sin imponer cuál, así que nunca responde solo con texto. Combinado con tool_choice {type: tool, name: "extract_metadata"} obliga a Claude a llamar a esa herramienta, lo que asegura que la extracción se ejecute antes del enriquecimiento; en las peticiones siguientes se vuelve a auto. La respuesta empieza con el tool_use, sin texto previo ni bloque de razonamiento, y el pensamiento extendido manual no es compatible con forzar herramientas.
- Las salidas estructuradas garantizan que los campos obligatorios estén presentes con su tipo, así que un campo obligatorio sin dato en la fuente lleva a inventar. Si la información puede faltar, declara el campo con un tipo que admita null (type: ["string", "null"] o anyOf con null) e indica cuándo usarlo. Las propiedades obligatorias salen antes que las opcionales.
- En el enum estricto solo deja elegir entre los valores listados, así que los casos que no encajan acaban forzados en la categoría más cercana. Añadir "unclear" para los casos ambiguos y "other" con un campo de detalle hace la taxonomía ampliable sin perder datos legibles por máquina. Las mayúsculas de los enum no están garantizadas: compáralos sin distinguirlas.

Implement validation, retry, and feedback loops for extraction quality

- Un reintento útil no repite la misma petición: incluye el documento original, la extracción fallida y el error de validación concreto, para que Claude relea la fuente y corrija contra un criterio. En bucles con herramientas, el fallo se devuelve como tool_result con is_error a true y un mensaje instructivo, y el número de reintentos debe estar limitado.
- Un reintento solo sirve si la causa está al alcance de Claude: formatos incorrectos o errores de estructura se corrigen con el error concreto, y una salida cortada (stop_reason max_tokens) con un max_tokens mayor. Si el dato solo está en un documento que no se envió, reintentar no lo hará aparecer: hay que anotar la fuente o pasar el caso a revisión humana. Si los descartes de los desarrolladores se registran con esa etiqueta, se ve qué patrones concentran los falsos positivos y se puede ajustar el prompt o en Code Review, REVIEW.md solo para ellos, midiendo después el efecto.
- En el campo detected_pattern en cada hallazgo estructurado, mejor como enum con "other" y detalle, etiqueta que regla produjo. Para validar extracciones conviene que el esquema recoja a la vez lo que dice el documento y lo que se decide de él. stated_total junto a calculated_total permite detectar facturas que no cuadran, y un booleano obligatorio conflict_detected señala datos incoherentes en la fuente. Así los casos dudosos van a revisión humana en lugar de corregirse en silencio.

Design efficient batch processing strategies

- La Message Batches API procesa peticiones de forma asíncrona, con más rendimiento y menor coste, pero un lote puede tardar hasta 24 horas. Por eso sirve para análisis nocturnos o semanales, evaluaciones y reclasificaciones masivas, y no para operaciones que bloquean una fuejía ni para respuestas que alguien espera: esas van por la Messages API síncrona.
- Con la Message Batches API, un plazo garantizado debe reservar el peor caso: la espera en cola hasta el siguiente en vivo hasta 24 horas de procesamiento y cualquier paso posterior. Así, con un SLA de 30 horas, hay que enviar al menos cada 6 horas (4 deja margen). Las peticiones que vuelven como expired pueden necesitar la síncrona para cumplir el plazo.
- Cada resultado de un lote lleva el custom_id de su petición y el orden no está garantizado, así que los fallos se identifican por custom_id y se reenvían solos en un lote nuevo, porque un lote enviado no se puede modificar. Las peticiones no válidas como un documento demasiado largo se corrigen antes (por ejemplo, fragmentándolo) las expiradas o con error de servidor se reenvían tal cual.
- Antes de lanzar un lote grande conviene afinar el prompt con una muestra representativa que refleje la distribución real e incluya casos límite, y calificarla de forma automática. También se recomienda probar una petición con la misma forma en la Messages API para evitar errores de validación. Así el lote acierta a la primera y se evitan reenvíos costosos.

Design multi-instance and multi-pass review architectures

- Un revisor que comparte la conversación del generador hereda su razonamiento y tiende a confirmarlo. Para una revisión independiente se usa otra instancia: una segunda sesión o un subagente que no sea fork, que empieza en un contexto limpio. Un fork o un subagente reanudado no sirven, porque heredan el historial. En Claude Code, tools: Read, Grep, Glob hace que el revisor no pueda editar.
- En revisiones grandes, un lío con contexto con muchos archivos diluye la atención y la profundidad. En Claude Code conviene revisar cada archivo en una pasada centrada (por ejemplo, en subagentes con contexto propio) para los problemas locales y añadir una pasada de integración aparte que examine interfaces, cambios y flujo de datos entre archivos, que ninguna pasada individual puede ver.
- Una pasada de verificación puede devolver, junto a cada hallazgo, un nivel de confianza en un campo propio de una salida estructurada con esquema. JSON, de modo que el código enrute los dudosos a revisión humana. Conviene exigir una cita o línea que respalde cada hallazgo, retirar los que no la tengan y contrastar los umbrales con casos etiquetados.

5. Context Management & Reliability

15–15%

Manage conversation context to preserve critical information across long interactions

- Resumir o compactar una conversación larga puede convertir importes, fechas, números de pedido y estados en frases vagas. Para evitarlo, el código extrae esos datos de las respuestas de las herramientas a un bloque de hechos explícito el esquema del estado y evitan mezclar datos entre problemas.
- Cuando un cliente plantea varios problemas en la misma sesión, conviene guardar cada uno como un registro estructurado (identificador de pedido, importe, estado) en una capa de contexto aparte que se envía en cada turno del caso que se incluye en cada petición, fuera del historial que se resume, en lugar de confiar en que el resumen o el modelo los conserven.
- Los resultados de herramientas se acumulan en el contexto y una consulta de pedido con decenas de campos gasta tokens que no aportan nada. Si controlas la herramienta, haz que devuelva solo los campos relevantes (por ejemplo con un parámetro `response_format` conciso o detallado); si no, en el Claude Agent SDK un hook `PostToolUse` puede sustituir la salida con `updatedToolOutput` antes de que Claude la vea.
- Con entradas largas que agregan muchos resultados, la precisión de recuperación baja a medida que crece el contexto y los hallazgos del medio se pierden con facilidad. Ayuda abrir la entrada agregada con un resumen de hallazgos clave, organizar el detalle en secciones etiquetadas (por ejemplo etiquetas XML por documento con su fuente) y situar los datos largos arriba y la pregunta y las instrucciones al final.
- El coordinador debe indicar a cada subagente un objetivo y un formato de salida: ese formato debe incluir, junto a cada hallazgo, metadatos como la fecha de los datos, la ubicación exacta de la fuente y el contexto metodológico. Sin ellos, el agente de síntesis no puede distinguir cifras antiguas de recientes ni explicar por qué dos fuentes discrepan, y las citas pierden precisión.

Design effective escalation and ambiguity resolution patterns

- Una instrucción vaga como escala los casos complejos produce escalados de más y de menos. Funciona mejor escribir en el system prompt criterios explícitos de escalado y añadir unos pocos ejemplos few-shot variados y canónicos, envueltos en etiquetas `example`, que muestren casos que se escalan y casos que el agente resuelve solo, en lugar de una larga lista de reglas para cada caso límite.
- Cuando un cliente pide expresamente hablar con una persona, el agente debe escalar en ese mismo turno, sin hacer antes consultas ni intentar resolver el caso. La precisión del escalado es una métrica clave de un agente de soporte, y el traspaso debe incluir un resumen estructurado de lo ya dicho en la conversación para que el cliente no tenga que repetirlo.
- La frustración por sí sola no es motivo de escalado: si el problema está al alcance del agente, debe reconocer el malestar y ofrecer la solución. Si el cliente reitera que prefiere a una persona, se escala sin volver a insistir. Escalar por sentimiento negativo satura la cola humana y baja la tasa de resolución sin intervención, dos métricas que la guía de soporte propone medir.
- Si la petición no dice nada o es ambigua sobre la petición concreta del cliente, el agente no debe improvisar una excepción ni denegar por defecto: debe escalar a una persona que tenga autoridad para decidir. El criterio de escalado es la laguna de la política, no la complejidad ni el importe, y conviene escribirlo de forma explícita junto con límites sobre lo que el agente no puede prometer.
- Si una búsqueda de cliente o de pedido devuelve varias coincidencias, el agente no debe elegir una por heurística (la más reciente, la primera de la lista): debe pedir un identificador adicional, como el número de pedido o el correo. La herramienta debe decir claro que hay varias coincidencias en lugar de devolver solo la primera, y el prompt debe indicar que no se adivinen parámetros que faltan.

Implement error propagation strategies across multi-agent systems

- Un error genérico como `search unavailable` o `failed` no deja al coordinador decidir nada. El subagente o la herramienta debe devolver un error estructurado: tipo de fallo, qué se intentó, resultados parciales y alternativas posibles. En la Claude API se envía en un bloque `tool_result` con `is_error` a `true` y un mensaje instructivo; en MCP, en el resultado con `isError`.
- Una consulta que no encuentra nada es un resultado válido: una consulta que no pudo ejecutarse (timeout, servicio caído, permisos) es un fallo de acceso. Si ambos llegan igual, el coordinador afirma ausencias falsas o reintenta sin motivo. El fallo debe marcarse como error (`is_error` en la Claude API, `isError` en MCP) y el vacío como éxito con cero resultados.
- Los fallos transitorios, como un timeout o un límite de peticiones momentáneo, se resuelven mejor dentro del subagente con reintentos acotados. Solo los errores que no puede resolver, como un permiso denegado o reintentos agotados, deben llegar al coordinador, acompañados de lo que se intentó y de los resultados parciales. Así el coordinador no se llena de ruido y conserva el trabajo ya hecho.
- Cuando algunas fuentes no estuvieron disponibles, el informe de síntesis no debe sonar completo. Conviene estructurarlo con anotaciones de cobertura por sección o hallazgo: qué está bien respaldado por varias fuentes, qué depende de una sola y qué áreas tienen lagunas por fuentes caídas. Dar permiso explícito para decir que falta información reduce las afirmaciones inventadas.

Manage context effectively in large codebase exploration

- En una exploración larga de código, conviene delegar preguntas concretas (localizar archivos de test, trazar dependencias de un flujo) en subagentes, como el Explore integrado de Claude Code. Cada uno trabaja en su propia ventana de contexto y devuelve solo un resumen, mientras la conversación principal conserva el plan. Un subagente no hereda el historial: lo que necesite debe ir en el prompt de delegación.
- En sesiones largas de exploración, el agente empieza a dar respuestas incoherentes y genéricas porque los hallazgos concretos quedan enterrados o se resumen. La toma de notas estructurada lo evita: el agente escribe en un archivo (por ejemplo `NOTES.md`) clases, rutas y decisiones a medida que las descubre y lo vuelve a leer antes de responder preguntas posteriores o tras una compactación.
- Entre fases de una exploración, el coordinador debe resumir los hallazgos clave de la fase terminada, guardarlos si hace falta en memoria externa y pasar ese resumen en el prompt inicial de los subagentes de la fase siguiente. Así empiezan con un contexto limpio pero informado, sin repetir trabajo ni heredar el historial completo, que es lo que haría un fork.
- Para recuperarse de un fallo sin empezar de cero, cada agente exporta su estado (tareas hechas, pendientes, rutas de resultados) a una ubicación conocida, y al reanudar el coordinador carga ese manifiesto e inyecta en el prompt de cada agente la parte que le toca. Conviene usar JSON con campos explícitos, porque el modelo tiende menos a reescribir JSON que Markdown, junto con git y notas de progreso.
- En Claude Code, `/compact` resume la conversación para liberar contexto sin abandonar la tarea, y admite instrucciones de enfoque, como `/compact` centra el resumen en el módulo de auth. `/clear` empieza una conversación vacía. Tras compactar se vuelven a cargar desde disco el `CLAUDE.md` raíz y las reglas sin `paths:`, así que lo que deba sobrevivir siempre conviene ponerlo ahí o en una sección Compact instructions.

Design human review workflows and confidence calibration

- Si las extracciones de confianza alta se aprueban sin revisión, nadie mide su tasa de error real ni ve aparecer fallos nuevos. Conviene enviar a revisión humana una muestra aleatoria de esas extracciones, estratificada por tipo de documento o proveedor, de forma continua. La revisión humana detecta casos límite que la evaluación automática no ve.
- Antes de reducir la revisión humana, mide la precisión con un conjunto etiquetado desglosado por tipo de documento y por campo: una cifra global alta puede ocultar un segmento muy por debajo del objetivo. Automatiza solo los segmentos que cumplen el umbral, mantén la revisión en los demás y vuelve a medirlos tras cada cambio. Las salidas estructuradas garantizan el esquema, no la exactitud de los valores.
- Pide al modelo una puntuación de confianza por campo en la salida estructurada, no una sola por documento, para enviar a revisión solo los valores dudosos. Esa puntuación no es una precisión: compárala con un conjunto de validación etiquetado y fija cada umbral donde la precisión observada alcance el objetivo. Las salidas estructuradas garantizan el formato, no que los valores sean correctos.
- Con revisores limitados, la cola humana debe recibir solo extracciones con un indicio concreto de error: campos por debajo de un umbral calibrado y documentos en los que el modelo señala valores ambiguos o contradictorios en la fuente. Ordena la cola por impacto. Un muestreo aleatorio o un umbral general demasiado alto desperdician capacidad, y pedir al modelo que elija un valor oculta el conflicto.

Preserve information provenance and handle uncertainty in multi-source synthesis

- La procedencia se pierde si los subagentes devuelven resúmenes sin estructura: pide que cada hallazgo sea una afirmación con su URL o nombre de documento y el fragmento que la respalda, y que la síntesis conserve esas asociaciones. Con la API, Citations devuelve el `cited_text` exacto y punteros validos al documento, no se combina con salidas estructuradas, así que pide una línea por campo y analízala en código.
- Un informe multifuente debe separar en secciones explícitas los hallazgos consolidados de los disputados, y conservar cómo describe cada fuente su resultado (estimación preliminar, piloto pequeño, encuesta auditada) y su método. Los subagentes deben señalar especulaciones y problemas de las fuentes en vez de presentarlo todo como hechos, y la síntesis no debe eliminar esas cautelas al resumir.
- Cuando las fuentes discrepan y el subagente no puede conciliarlas, debe terminar el análisis e incluir todos los valores con su fuente y una anotación explícita del conflicto. La decisión corresponde al coordinador, que ve los resultados de todos los subagentes, antes de pasar nada a la síntesis. Elegir un valor en silencio, hacer medias o detenerse en el primer conflicto oculta información o deja el trabajo a medias.
- Una cifra sin fecha no se puede interpretar en el tiempo: los subagentes deben incluir en cada hallazgo la fecha de publicación y el periodo de recogida de datos, y el prompt debe dar la fecha actual. Así la síntesis distingue referencias antiguas de datos recientes y aplica la recencia al resolver conflictos. Al pasar varios documentos, añade la fecha como metadato junto al origen de cada uno.
- En los informes de síntesis, cada tipo de contenido tiene su forma: cifras financieras comparables en tablas, noticias en prosa y hallazgos técnicos en listas estructuradas. Indícalo en el prompt por tipo de contenido y en positivo («escribe esta sección en párrafos de prosa») en lugar de prohibiciones generales como «no uses markdown», que eliminan también las tablas necesarias.