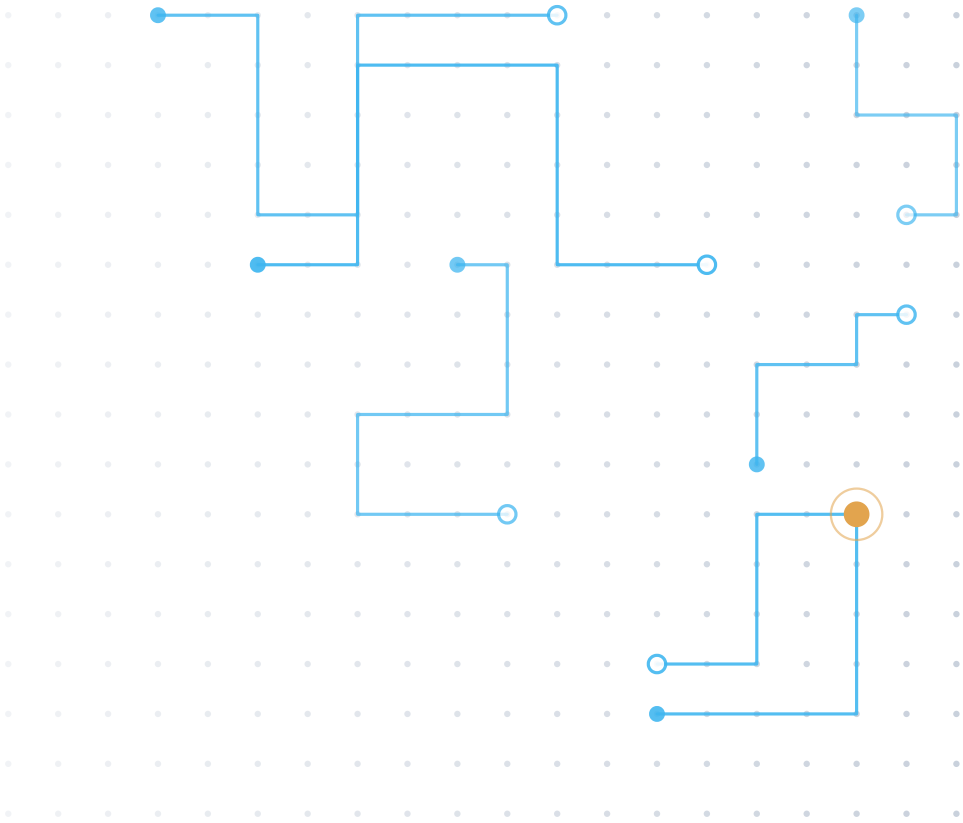


GUÍA PRÁCTICA

Claude Code en un servidor: tareas programadas y límites

Deja a Claude Code trabajando de noche sin que se coma la máquina ni el presupuesto

AUTOR
Mario Chasco · Flowara AI

PARA
Equipos técnicos que ya usan Claude Code y quieren dejarlo trabajando sin supervisión

LECTURA
9 min

EDICIÓN
v2.0 · noviembre 2026

ANTES DE EMPEZAR

Qué vas a sacar de esta guía

01 Distinguir las tareas que pueden quedarse solas de noche de las que necesitan a una persona delante.

02 Lanzar cada tarea larga dentro de una jaula de recursos que protege la máquina y el editor.

03 Montar trabajos por lotes que se reanudan tras un corte sin repetir envíos.

04 Gastar menos tokens leyendo por índice y dando cada paso al modelo que le corresponde.

05 Recibir cada mañana un informe en el que ningún fallo pasa en silencio.

CONTENIDO

01 El problema: la lavadora, no el grifo abierto
02 Tareas programadas dentro de una jaula
03 Menos herramientas, más ficheros
04 Trabajos reanudables, por lotes y de uno en uno
05 Presupuesto de tokens: leer menos, decidir mejor
06 Supervisión: que nada falle en silencio
07 Ejemplo: revisión nocturna para una agencia de viajes
08 Lista antes de dejarlo solo

Para: Equipos técnicos que ya usan Claude Code y quieren dejarlo trabajando sin supervisión · 9 min de lectura

01

El problema: la lavadora, no el grifo abierto

Dejar la lavadora puesta antes de dormir es razonable: tiene un programa, un límite de agua y se para sola. Dejar el grifo abierto no lo es, aunque el agua sea la misma. Con Claude Code en un servidor pasa igual. Con programa y límites te adelanta trabajo mientras duermes; sin ellos, es un proceso que se desborda.
Lo viví en mi propio servidor, que comparten varias sesiones de trabajo a la vez. Una tarea masiva llenó la memoria, el sistema empezó a cerrar procesos para sobrevivir y se llevó por delante el editor con trabajo sin guardar. El modelo no falló; fallaron los límites. Desde entonces cada tarea larga corre con las reglas de esta guía.

No todo vale para dejarlo solo. Mi criterio es sencillo: de noche, solo lo que deja un resultado revisable por la mañana y no tiene efectos difíciles de deshacer.

TAREA	SOLA DE NOCHE	POR QUÉ
Resúmenes y fichas de documentos	Verde	Deja un borrador que se revisa
Reindexar una base de conocimiento	Verde	Se puede repetir sin efectos
Revisar enlaces, formatos y fechas	Verde	Solo informa, no cambia nada
Preparar borradores de respuesta	Ámbar	Los envía una persona por la mañana
Responder a clientes sin revisión	Rojo	Un error llega directo al cliente
Desplegar a producción	Rojo	Difícil de deshacer a las cuatro de la mañana
Cualquier cosa con dinero o permisos	Rojo	El error sale caro

EN UNA FRASE

Sin nadie mirando, una tarea necesita límites que no pueda saltarse y un resultado que alguien revise por la mañana.

02

Tareas programadas dentro de una jaula

Las tareas recurrentes se programan con lo de siempre: un temporizador de *systemd* o una línea de *cron*. Lo que importa es dónde corren. Cada tarea larga va dentro de una jaula de recursos, lo que Linux llama un *cgroup*, con límites que no puede saltarse. Así la lanzo yo:

BASH

```
# Tarea larga en una jaula de recursos, fuera del editor
systemd-run --user --collect --unit=informe-nocturno \
  -p Nice=19 \
  -p CPUSchedulingPolicy=idle \
  -p IOSchedulingClass=idle \
  -p CPUQuota=50% \
  -p MemoryMax=1400M \
  -p OOMScoreAdjust=500 \
  /srv/jobs/informe-nocturno/run.sh
```

Cada línea tiene un motivo. La prioridad mínima y la planificación *idle* hacen que la tarea use solo lo que nadie más está usando. La cuota pone techo al procesador; el máximo de memoria corta la tarea antes de que asfixie la máquina. La puntuación alta ante falta de memoria asegura que, si el sistema tiene que matar algo, empiece por ella y no por tu editor. Y como corre como servicio aparte, sobrevive aunque cierres la sesión.

Los valores son un punto de partida. Ajústalos a tu máquina, pero no quites ninguno.

> *La tarea puede fallar; la máquina, nunca.*
>
> LA REGLA QUE SIGO EN MI SERVIDOR

03

Menos herramientas, más ficheros

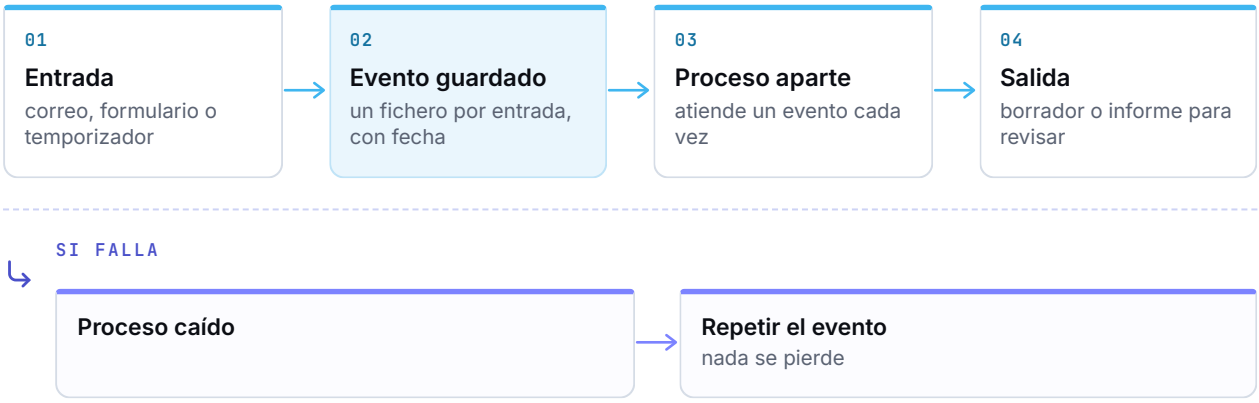
Una tarea nocturna no necesita acceso a todo. Cuantas menos herramientas tenga a mano, menos cosas pueden salir mal. Prefiero un script pequeño por servicio, con los permisos justos para lo que hace: el que lee el calendario no puede escribir en él, y el que redacta correos los deja en una carpeta en vez de enviarlos.

Las claves van fuera del prompt. Viven en el entorno del servicio o en un gestor de secretos, nunca en el texto que lee el modelo ni en el repositorio.

> ⚠ **AVISO**
>
> El prompt no es una capa de permisos. Escribir «no borres nada» ayuda, pero no impide nada. Lo que de verdad limita es una clave con alcance reducido: si la clave solo permite leer, la tarea solo puede leer, diga lo que diga el prompt.

Para las tareas que reaccionan a algo, como un correo que llega o un formulario enviado, uso siempre el mismo patrón. La entrada se guarda como evento antes de procesarla, y un proceso aparte la recoge cuando le toca:

FIGURA 1 **El patrón de cada tarea en mi servidor**



El evento guardado hace dos trabajos. Es la auditoría, porque siempre puedes ver qué entró y cuándo. Y permite repetir: si el proceso falla, el evento sigue ahí y se vuelve a procesar sin pedir nada a nadie.

04 Trabajos reanudables, por lotes y de uno en uno

Una tarea nocturna se va a interrumpir: un reinicio, un corte de red, el límite de memoria haciendo su trabajo. La pregunta es cuánto se pierde, y la respuesta correcta es casi nada.

FIGURA 2 **Un lote que sobrevive a una interrupción**



Lo consiguen tres hábitos:

01 **Estado por elemento.** Cada documento o ficha procesada se marca como hecha en un archivo de estado. Al reiniciar, la tarea lo lee y sigue donde lo dejó.

02 **Lotes pequeños con pausas.** Mejor diez lotes de cincuenta que uno de quinientos. Cada pausa deja respirar a la máquina y a los límites de uso del modelo.

03 **Un trabajo pesado a la vez.** Antes de lanzar miro la memoria libre y la carga con `free -h` y `uptime`. Si hay poca memoria, lote más pequeño o esperar.

> ⚠ **CUIDADO**
>
> Reanudable no significa que pueda repetir efectos. Si un paso envía algo, guarda que ya lo envió antes de pasar al siguiente. Si no, un reinicio a mitad de lote manda el mismo correo dos veces.

EN UNA FRASE

Una interrupción debe costar un elemento, no una noche.

05

Presupuesto de tokens: leer menos, decidir mejor

Sin nadie mirando, el gasto de tokens también se desborda en silencio. Estas son las prácticas que lo mantienen a raya en mis tareas nocturnas. **El modelo adecuado para cada paso.** Clasificar, resumir o comprobar un formato lo hace bien un modelo pequeño y rápido como Haiku 5.5. Redactar lo rutinario encaja en Sonnet 5.5. Opus 5.5 se reserva para lo que necesita criterio, como decidir qué cambio se propone o detectar algo que no cuadra. Una buena tarea nocturna pasa casi todo por el pequeño y solo sube al grande lo que el pequeño ha marcado como dudoso. Lo explico con detalle en la guía *El modelo adecuado para cada tarea*.

Contexto pequeño. Cada llamada lleva solo lo necesario: el elemento actual, las reglas y el formato de salida. Nunca el historial entero ni el archivo completo «por si acaso».

Índice, búsqueda, fragmento. Primero se consulta un índice, después se busca y al final se carga solo el fragmento que importa. Leer un registro de diez mil líneas para encontrar tres errores es la forma más cara de hacerlo; una búsqueda y un recuento dan lo mismo por casi nada. Si una tarea necesita cargar un archivo entero, pregúntate primero si le falta un índice.

EN UNA FRASE

Paga por criterio, no por lectura.

06

Supervisión: que nada falle en silencio

Sin supervisión no significa sin vigilancia. La vigilancia la hace el propio sistema.

Hooks que bloquean malos hábitos. Claude Code permite ejecutar pequeños programas, los *hooks*, en momentos clave de la sesión. El que más me ha servido se ejecuta al terminar cada turno: si el último mensaje promete trabajo («ahora empiezo…») pero no se ha lanzado nada, impide cerrar el turno. Obliga a lanzarlo o a decir con claridad que no ha empezado. Nació de un error real en mi servidor: una sesión anunció un trabajo, no lo lanzó y la máquina estuvo horas parada.

Contadores. Cada tarea registra cuántos elementos ha procesado, cuántos han fallado y cuántos ha saltado. Un fallo que no suma en un contador es un fallo que nadie verá.

Informe de la mañana. Un resumen breve con lo que se ejecutó, lo que tardó y lo que espera revisión. Incluye un canario, una tarea mínima cuya ausencia delata que algo se ha parado.

> ✦ REGLA
>
> Cada error del informe se convierte en una regla escrita o en un hook que impida repetirlo.

EN UNA FRASE

Un fallo que no se cuenta ni se informa es un fallo que se repite.

07

Ejemplo: revisión nocturna para una agencia de viajes

Una agencia de viajes tiene 300 fichas de destinos en su web. Quiere saber cada mañana cuáles tienen enlaces rotos, fechas pasadas o textos viejos, sin que el sistema corrija nada por su cuenta. Así lo montaría:

FIGURA 3 **La revisión nocturna, paso a paso**

1. **Programación**
   Un temporizador lanza la tarea de madrugada, en una jaula con techo de procesador y de memoria.
2. **Lotes**
   La tarea revisa las fichas de 25 en 25, con una pausa entre lotes, y marca cada ficha en su archivo de estado.
3. **Reparto**
   Enlaces y fechas los comprueba código, sin IA. Haiku 5.5 lee cada ficha y marca las dudosas; solo esas pasan a Opus 5.5, que redacta una propuesta.
4. **Borradores**
   Nada se publica. Las propuestas quedan en una carpeta de revisión.
5. **Interrupción**
   A mitad de la noche el servidor se reinicia. La tarea rearranca, lee el estado y sigue en la ficha 176.
6. **Informe**
   Por la mañana llega el resumen, con cada fallo explicado y una persona con nombre que lo lee.

El informe de esa mañana cabe en tres cifras:

300	12	5
fichas revisadas durante la noche	enlaces rotos detectados por código	propuestas de texto para revisar

Fuente: informe de una mañana en la agencia (ejemplo sintético)

La persona que lo lee revisa las cinco propuestas y decide cuáles publica. Los enlaces rotos los arregla después, con la lista delante. Ningún fallo quedó sin explicar.

EN UNA FRASE

La máquina trabaja de noche; la persona decide por la mañana.

08

Lista antes de dejarlo solo

Repasa esta lista antes de programar cualquier tarea desatendida. Si una casilla queda vacía, la tarea todavía no está lista para la noche.

Antes de dejar una tarea sola

14 PUNTOS

- [ ] Objetivo escrito y resultado revisable
  lo que se mira por la mañana
- [ ] Sin despliegues, dinero ni respuestas a clientes
  salvo con revisión
- [ ] Se lanza con temporizador o cron
  nunca desde una sesión abierta
- [ ] Corre en una jaula de recursos
  cuota, memoria, prioridad y puntuación
- [ ] Corre fuera del proceso del editor
  un fallo no se lleva tu trabajo
- [ ] Memoria y carga comprobadas antes
  un solo trabajo pesado a la vez
- [ ] Claves con alcance reducido
  nunca dentro del prompt
- [ ] Cada entrada se guarda como evento
  se puede auditar y repetir
- [ ] Estado por elemento y lotes pequeños
  se reanuda sin repetir envíos
- [ ] Cada paso con el modelo más pequeño que lo hace bien
- [ ] Lee por índice, búsqueda y fragmento
  nunca archivos enteros por defecto
- [ ] Hooks contra los malos hábitos conocidos
- [ ] Contadores de procesados, fallidos y saltados
- [ ] Informe de la mañana con canario
  y una persona con nombre que lo lee

SIGUIENTE PASO

Diagnóstico gratuito (20 min)

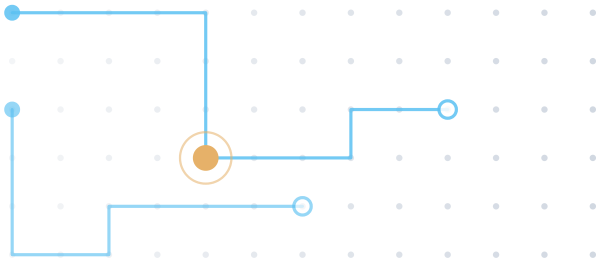
Si quieres ver qué parte de tu negocio se puede automatizar con el riesgo bajo control, reserva un diagnóstico gratuito de 20 minutos.

Reservar diagnóstico ›

flowaraai.com/es/contacto

RECURSOS RELACIONADOS

- Kit de hooks y agentes para Claude Code ↗
- Base de conocimiento de IA que se actualiza sola ↗



Mario Chasco · Flowara AI

Sistemas de IA en producción para negocios pequeños.

flowaraai.com
github.com/mariochasco78557
youtube.com/@mariochasco_AI
linkedin.com/in/mario-adrian-chasco-frauca

Puedes compartir esta guía tal cual, citando flowaraai.com.