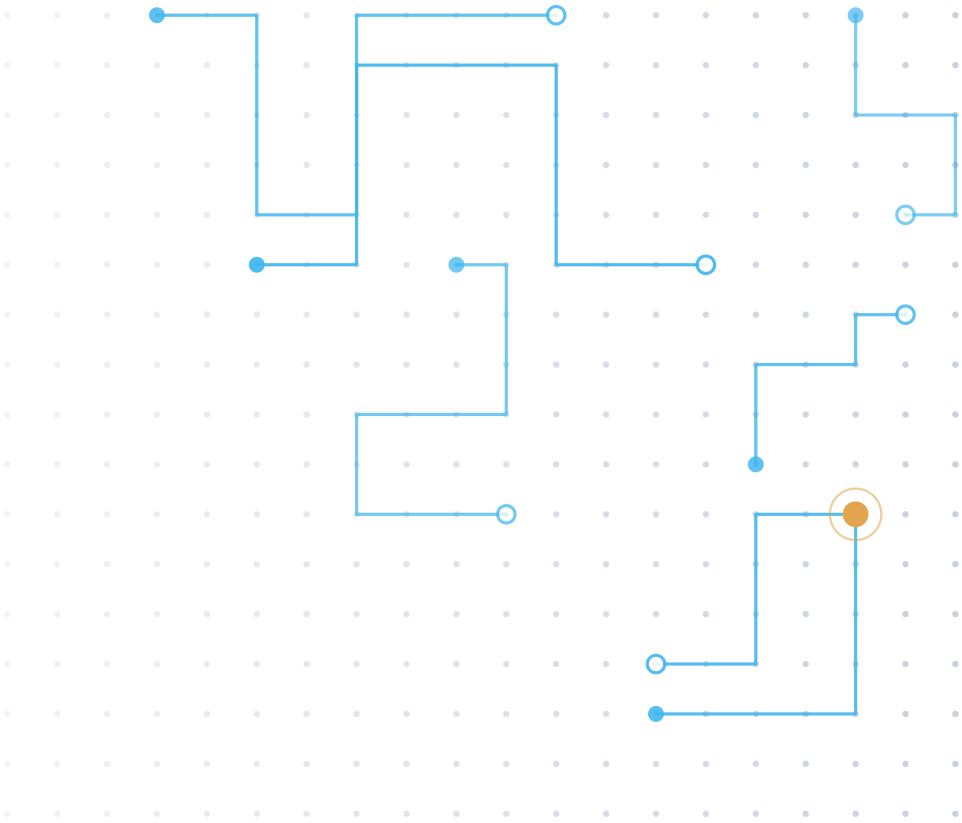


PRACTICAL GUIDE

Claude Code on a server: scheduled jobs and limits

Leave Claude Code working overnight without it eating the machine or the budget

AUTHOR
Mario Chasco · Flowara AI

FOR
Technical teams already using Claude Code who want it to run unattended

READING
9 min

EDITION
v2.0 · November 2026

BEFORE YOU START

What you will get from this guide

01 Tell apart the jobs that can run alone overnight and the ones that need a person watching.

02 Launch every long job inside a resource cage that protects the machine and your editor.

03 Build batch jobs that pick up after an interruption without sending anything twice.

04 Spend fewer tokens by reading through an index and giving each step to the right model.

05 Get a morning report in which no failure goes unnoticed.

CONTENTS

01 The problem: a washing machine, not an open tap
02 Scheduled jobs inside a cage
03 Fewer tools, more files
04 Resumable jobs, in batches, one at a time
05 Token budget: read less, decide better
06 Supervision: nothing fails silently
07 Example: an overnight review for a travel agency
08 Checklist before leaving it alone

For: Technical teams already using Claude Code who want it to run unattended · 9 min read

01

The problem: a washing machine, not an open tap

Leaving the washing machine on before bed is sensible. It has a programme and a water limit, and it stops by itself. Leaving a tap running is not sensible, even though the water is the same. Claude Code on a server works the same way. With a programme and limits it gets work done while you sleep; without them, it is a process that overflows.
I have been through this on my own server, which several working sessions share at the same time. A bulk job filled the memory, the system started killing processes to survive, and it took down my editor along with unsaved work. The model did not fail; the limits did. Since then every long job on that server follows the rules in this guide.

Not everything should be left alone. My rule of thumb is simple: overnight, only jobs that leave a result someone can review in the morning and that have no effects that are hard to undo.

JOB	ALONE OVERNIGHT	WHY
Document summaries and notes	Green	Leaves a draft that gets reviewed
Reindexing a knowledge base	Green	Can be repeated with no side effects
Checking links, formats and dates	Green	Only reports, changes nothing
Preparing reply drafts	Amber	A person sends them in the morning
Replying to customers without review	Red	A mistake goes straight to the customer
Deploying to production	Red	Hard to undo at four in the morning
Anything involving money or permissions	Red	Mistakes are expensive

IN ONE LINE

With nobody watching, a job needs limits it cannot get around and a result that someone reviews in the morning.

02

Scheduled jobs inside a cage

Recurring jobs are scheduled with the usual tools: a *systemd* timer or a *cron* line. What matters is where they run. Every long job goes inside a resource cage, which Linux calls a *cgroup*, with limits the job cannot get around. This is how I launch one:

```bash
# Long job inside a resource cage, outside the editor
systemd-run --user --collect --unit=nightly-report \
  -p Nice=19 \
  -p CPUSchedulingPolicy=idle \
  -p IOSchedulingClass=idle \
  -p CPUQuota=50% \
  -p MemoryMax=1400M \
  -p OOMScoreAdjust=500 \
  /srv/jobs/nightly-report/run.sh
```

Each line is there for a reason. Lowest priority and *idle* scheduling mean the job only uses what nobody else is using. The quota puts a ceiling on the processor; the memory maximum stops the job before it can choke the machine. The high out-of-memory score means that if the system has to kill something, it starts with this job and not with your editor. And because it runs as a separate service, it keeps going after you close your session.

Treat these values as a starting point. Adjust them to your machine, but keep all of them.

> *The job may fail; the machine, never.*
>
> THE RULE I FOLLOW ON MY SERVER

03

Fewer tools, more files

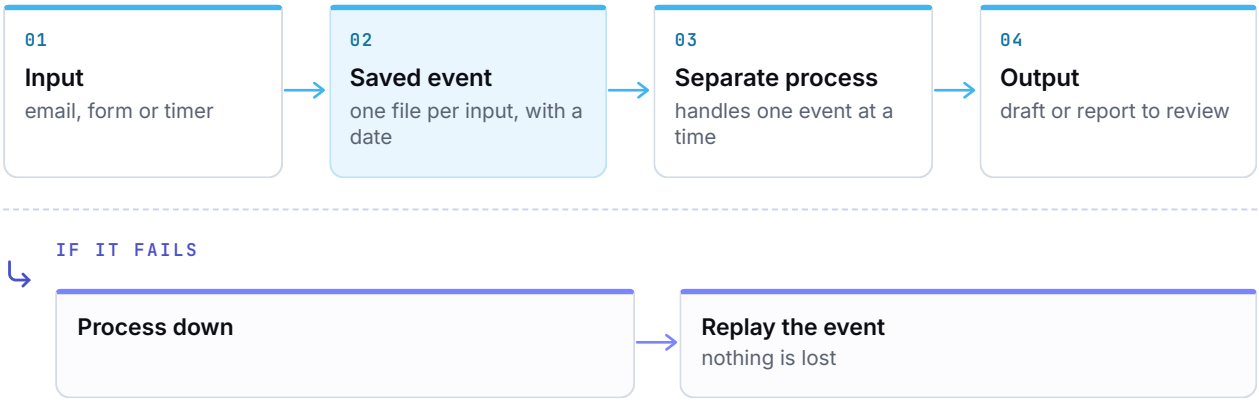
An overnight job does not need access to everything. The fewer tools it has within reach, the fewer things can go wrong. I prefer one small script per service, with only the permissions it needs: the one that reads the calendar cannot write to it, and the one that drafts emails leaves them in a folder instead of sending them.

Keys stay out of the prompt. They live in the service environment or in a secrets manager, never in the text the model reads and never in the repository.

> ⚠ **WARNING**
>
> The prompt is not a permissions layer. Writing "do not delete anything" helps, but it stops nothing. What actually limits a job is a key with a narrow scope: if the key can only read, the job can only read, whatever the prompt says.

For jobs that react to something, such as an incoming email or a submitted form, I always use the same pattern. The input is saved as an event before anything processes it, and a separate process picks it up when its turn comes:

FIGURE 1 **The pattern behind every job on my server**

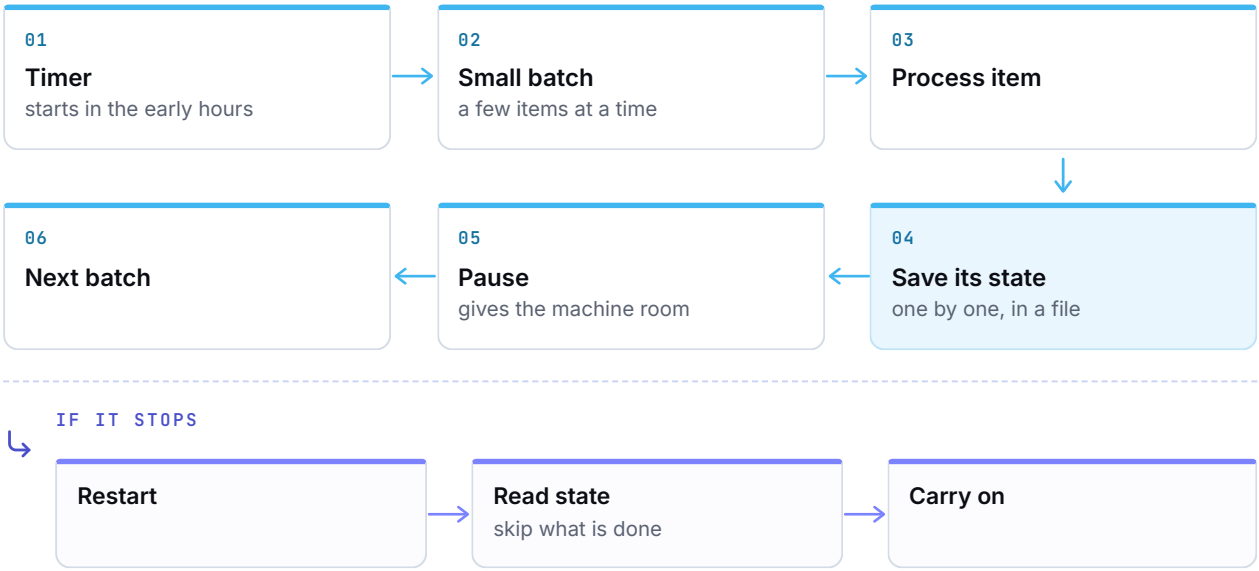


The saved event does two jobs. It is the audit trail, because you can always see what came in and when. It also lets you replay: if the process fails, the event is still there and gets processed again without asking anyone for anything.

04 Resumable jobs, in batches, one at a time

An overnight job will get interrupted: a reboot, a network cut, the memory limit doing its job. The real question is how much you lose, and the right answer is almost nothing.

FIGURE 2 **A batch that survives an interruption**



Three habits get you there:

01 **State per item.** Every document or record processed is marked as done in a state file. On restart, the job reads it and picks up where it left off.

02 **Small batches with pauses.** Ten batches of fifty are better than one of five hundred. Each pause gives the machine room, and the model's usage limits too.

03 **One heavy job at a time.** Before launching I check free memory and load with `free -h` and `uptime`. If memory is short, I use a smaller batch or wait.

> ⚠ CAUTION
>
> Resumable does not mean it may repeat side effects. If a step sends something, record that it was sent before moving on. Otherwise a reboot halfway through a batch sends the same email twice.

IN ONE LINE

An interruption should cost one item, not one night.

05

Token budget: read less, decide better

With nobody watching, token spend also overflows quietly. These are the practices that keep it under control in my overnight jobs.

The right model for each step. Classifying, summarising or checking a format is done well by a small, fast model such as Haiku 5.5. Routine drafting fits Sonnet 5.5. Opus 5.5 is kept for work that needs judgment, such as deciding which change to suggest or spotting something that does not add up. A good overnight job sends almost everything through the small model and only passes up what the small model has flagged as doubtful. I cover this in detail in the guide *The right model for each task*.

Small context. Each call carries only what it needs: the current item, the rules and the output format. Never the whole history, and never the full file "just in case".

Index, search, fragment. First look at an index, then search, and only then load the fragment that matters. Reading a ten-thousand-line log to find three errors is the most expensive way to do it; a search and a count give you the same answer for almost nothing. If a job needs to load a whole file, first ask whether it is missing an index.

IN ONE LINE

Pay for judgment, not for reading.

06

Supervision: nothing fails silently

Unattended does not mean unwatched. The system does its own watching.

Hooks that block bad habits. Claude Code can run small programs, called *hooks*, at key moments of a session. The one that has helped me most runs at the end of every turn. If the last message promises work ("starting on it now...") but nothing has been launched, the hook stops the turn from closing. The session has to launch the work or say clearly that it has not started. It came from a real mistake on my server: a session announced a job, never launched it, and the machine sat idle for hours.

Counters. Every job records how many items it processed, how many failed and how many it skipped. A failure that does not add to a counter is a failure nobody will see.

A morning report. A short summary of what ran, how long it took and what is waiting for review. It includes a canary, a tiny job whose absence tells you something has stopped.

> ✦ RULE
>
> Every error in the report becomes a written rule or a hook that stops it happening again.

IN ONE LINE

A failure that is neither counted nor reported is a failure that repeats.

07

Example: an overnight review for a travel agency

A travel agency has 300 destination pages on its website. It wants to know every morning which pages have broken links, past dates or stale copy, without the system fixing anything on its own. This is how I would set it up:

FIGURE 3 **The overnight review, step by step**

1. **Scheduling**
   A timer launches the job in the early hours, inside a cage with a processor and memory ceiling.
2. **Batches**
   The job reviews the pages 25 at a time, pausing between batches, and marks each page in its state file.
3. **Model split**
   Plain code checks links and dates, with no AI. Haiku 5.5 reads each page and flags the doubtful ones; only those go to Opus 5.5, which drafts a suggestion.
4. **Drafts**
   Nothing gets published. Suggestions land in a review folder.
5. **Interruption**
   Halfway through the night the server reboots. The job restarts, reads its state and resumes at page 176.
6. **Report**
   In the morning the summary arrives, with every failure explained and a named person who reads it.

That morning's report fits in three figures:

300	12	5
pages reviewed overnight	broken links found by plain code	copy suggestions waiting for review

Source: one morning's report at the agency (synthetic example)

The person who reads it goes through the five suggestions and decides which ones to publish. The broken links get fixed afterwards, with the list in front of them. No failure was left unexplained.

IN ONE LINE

The machine works at night; the person decides in the morning.

08

Checklist before leaving it alone

Go through this list before you schedule any unattended job. If a box stays empty, the job is not ready for the night yet.

Before leaving a job alone

14 ITEMS

- [ ] Written goal and a reviewable result
  what you check in the morning
- [ ] No deploys, money or customer replies
  unless someone reviews them
- [ ] Launched by a timer or cron
  never from an open session
- [ ] Runs inside a resource cage
  quota, memory, priority and score
- [ ] Runs outside the editor's process tree
  a failure cannot take your work
- [ ] Memory and load checked first
  one heavy job at a time
- [ ] Keys with a narrow scope
  never inside the prompt
- [ ] Every input saved as an event
  it can be audited and replayed
- [ ] State per item and small batches
  resumes without sending twice
- [ ] Each step on the smallest model that does it well
- [ ] Reads by index, search and fragment
  never whole files by default
- [ ] Hooks against the bad habits you know
- [ ] Counters for processed, failed and skipped items
- [ ] Morning report with a canary
  and a named person who reads it

NEXT STEP

Free diagnostic (20 min)

If you want to see which part of your business can be automated with the risk under control, book a free 20-minute diagnostic.

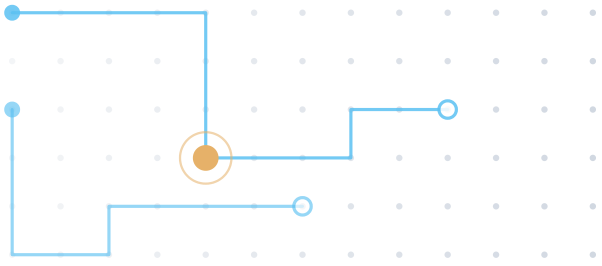
Book a diagnostic ›

flowaraai.com/en/contact

RELATED RESOURCES

Hooks and agents toolkit for Claude Code ↗

Self-updating AI knowledge base ↗



Mario Chasco · Flowara AI

AI systems in production for small businesses.

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

Share this guide as is, crediting flowaraai.com.