Claude Code Routines

This guide covers Claude Code's new **Routines** feature: what it is, how to set it up, the gotchas Nate ran into, and everything you need to migrate your automations to the cloud.

By: Nate Herk

I. What Are Routines?

Routines are saved Claude Code tasks that run on **Anthropic's cloud servers**, on a schedule, when an API is called, or when something happens on GitHub, even when your laptop is closed.

Announced April 14 in research preview: you configure a routine once (essentially a prompt), and it can run on a schedule, from an API call, or in response to a GitHub event, all on Anthropic's web infrastructure.

Status: Research preview. Behavior, limits, and API surface may change. Available on Pro, Max, Team, and Enterprise plans with Claude Code on the web enabled.

Where you can create them:

- The terminal (via scheduled remote agents)
- claude.ai/code (web)
- The Claude Code desktop app (what Nate demos in the video)

Analogy: It's like hiring a contractor who clones your repo, works on a separate branch, pushes it, and leaves. You review and decide what to merge. Or, think of a routine as you typing a prompt into Claude Code, except someone else is sitting at your laptop typing it for you.

II. How It Works

- You define a routine: a **prompt + one or more GitHub repos + triggers + connectors**
- Each run **clones your repo fresh**, does the work in an isolated cloud environment, and **destroys the environment after**
- If Claude makes changes, it pushes them to a **claude/-prefixed branch** on your real GitHub repo
- Every run creates a **session on claude.ai** where you can review what Claude did, continue the conversation, or create a PR

- Actions happen as **YOU** (commits carry your GitHub user, Slack messages use your account)

Because routines work off a cloned repo, **CLAUDE.md is read automatically every run**, which is both powerful and a reason to think carefully about which repo you point a routine at.

III. Setting Up a New Routine

In the desktop app, click **New Task** and choose local or remote. Configuration includes:

- **Name:** what the routine is called
- **Prompt:** what Claude should do (this IS the routine)
- **Model:** pick any available Claude model
- **Repository:** the GitHub repo to clone
- **Cloud environment:** env vars, network access, setup script
- **Cadence:** hourly, daily, weekdays (minimum interval: **1 hour**)
- **Connectors:** Slack, Gmail, Linear, etc.
- **Permissions:** how autonomously Claude should act

> **Critical mindset:** Routines are meant to be **one-shot prompts**. You're not around to answer questions. Write prompts specific enough that Claude can complete the job without stopping to ask anything, otherwise the automation defeats its own purpose.

IV. CLAUDE.md Makes Routines Smart

Routines clone your repo fresh every run, which means they **read your CLAUDE.md automatically**.

This is how you go from generic AI runs to context-aware ones:

- Coding standards, architecture notes, project conventions, all loaded before Claude writes a line
- Include instructions like *"always run tests before pushing"* or *"never modify the auth module without flagging it"*
- The better your CLAUDE.md, the better every routine performs

Tip: If a routine covers a specific subfolder, drop a CLAUDE.md inside that subfolder for more targeted instructions.

Counter-tip from the video: If you have a massive project (Nate's "Herk 2" example) with tons of context in CLAUDE.md, you may *not* want to point a routine at it. You'll burn tokens on context that's irrelevant to that automation. **Consider creating a dedicated GitHub repo per routine** so the context stays lean.

V. The Three Trigger Types

Trigger	How it fires	Example
Schedule	Cron-based (hourly, daily, weekly). Min interval: 1 hour.	"Every weeknight at 9pm, triage new issues"
API	POST to a dedicated endpoint with a bearer token	"When Sentry fires an alert, Claude diagnoses and opens a fix PR"
GitHub	Reacts to repo events (PRs, pushes, issues, releases, workflow runs, etc.)	"On every new PR, run my team's review checklist"

- You can **combine multiple triggers** on one routine (e.g., runs nightly AND on every new PR)
- GitHub triggers support **18+ event types** including PRs, pushes, issues, releases, check runs, workflow runs, discussions, merge queue entries
- **PR filters available:** author, title, body, base/head branch, labels, draft status, merged status, from fork

---

VI. Routines vs. Existing Options

	Routines (NEW)	Desktop Scheduled Tasks	/loop
Runs on	Anthropic cloud	Your machine	Your machine
Needs machine on	**No**	Yes	Yes
Needs open session	No	No	Yes
Survives restarts	Yes	Yes	No
Local file access	No (fresh clone)	Yes	Yes

Permission prompts	None (fully autonomous)	Configurable	Inherits from session
Min interval	1 hour	1 minute	1 minute

> **Important:** Existing Desktop tasks do **NOT** migrate. They're a completely separate system. To move something to the cloud, you recreate it as a new routine manually.

VII. Environments: The Big Gotcha

Your .env is .gitignored, so the fresh clone won't have it. Routines solve this with **Environments**:

- **Environment variables:** put your API keys here (ClickUp, YouTube, whatever). Claude reads them like normal env vars during the run.
- **Network access:** controls what the routine can reach on the internet
- **Setup script:** install commands that run before each session (npm install, pip install, etc.)

You configure environments at claude.ai *before* creating the routine, then select one per routine.

The .env Gotcha (Nate's YouTube Example)

Nate's first test was a routine that fetched 50 YouTube comments via the YouTube Data API. **It failed the first time.** Why?

His scripts normally read API keys from .env, and CLAUDE.md reinforces that. But in a routine, **there is no .env file**, only the env vars you set in the cloud environment. Claude defaulted to looking for .env and errored out.

The fix: Explicitly tell the routine where to look. Nate updated his prompt to:

> *"My YouTube API key is available as an environment variable. Use it directly from the environment. Don't look for a .env."*

It worked on the next run.

General fix: Make your scripts read environment variables directly with `os.environ["KEY_NAME"]` instead of relying on .env.

VIII. Network Access: Trusted vs. Full

Mode	What it does
Trusted (default)	Can only reach known/vetted services (GitHub, Anthropic, configured connectors). Whitelist approach.
Full	Can make outbound requests to anything on the internet. No restrictions.
None	No network
Custom	Allow specific domains

If your routine needs to hit external APIs (ClickUp, YouTube, Google Workspace, etc.), you need **Full**. Trusted will block those requests. Nate hit this exact wall when his ClickUp test failed on Trusted and only worked after switching to Full.

Risk of Full: If Claude reads malicious content during a run (a crafted PR description, compromised dependency), it could theoretically be tricked into sending data to an external server. Trusted would block that outbound request.

Practical risk for private repos where you control the inputs: very low. This matters more if you're processing untrusted input like public PRs from strangers.

IX. Connectors

Routines **can't use your local MCP servers**. Instead, they use **Connectors**, Anthropic's managed integrations that run in the cloud.

Available connectors at launch:

- **Slack:** read/post messages, respond to threads
- **Linear:** create/update issues, read project state
- **Jira:** ticket management
- **Google Drive:** read/write docs and files
- **GitHub** (built-in): PRs, issues, code changes

You attach connectors when creating a routine. Each connector authenticates through your account, so **actions happen as you**.

> **Nate's tip:** If you just need to send a notification, adding a Slack connector is often easier than wiring up a custom API call.

X. What Won't Work in Routines

Not everything belongs in a routine. These will fail:

- **Anything needing a persistent browser session:** Playwright/Puppeteer scripts requiring login cookies or session state. The environment is destroyed after each run, so there's no saved browser state. **Nate tested this with his Skool community automation and it doesn't work.** The clone has no access to local cookies from previous sessions. Workaround: use an endpoint with cookie/header/API-key auth instead of browser automation.
- **Local-only services:** Can't reach localhost databases, local APIs, or services on your machine
- **Resource-heavy processes:** Each run gets 4 vCPUs, 16 GB RAM, 30 GB disk. Exceed memory and the session is terminated.
- **Stateful workflows:** No state carries between runs. If step 2 depends on something step 1 saved to disk yesterday, it won't be there.

Rule of thumb: If it's local, or if Claude Code can't reach it via your GitHub repo or an API, it won't work.

Exception: If a routine modifies your codebase or does a review, the changes persist via the claude/ branch or session output, they're not destroyed with the environment.

XI. Security Details

- **Branch safety:** By default, routines can only push to claude/-prefixed branches. There's a per-repo "Allow unrestricted branch pushes" toggle (don't use it unless you must).
- **API trigger protection:** Each routine gets its own bearer token. Tokens shown once, can be regenerated or revoked.
- **If a token leaks:** Someone could spam-trigger runs and burn your daily cap. Revoke and regenerate immediately.
- **Everything runs as YOU:** Commits, PRs, Slack messages, Linear tickets, all carry your identity. If a routine posts to Slack at 3am with garbage, your name is on it. Test thoroughly before connecting routines to communication tools.
- **Don't put .env in your repo** to "solve" the env var problem. Even in a private repo, history persists and collaborators get exposed. Use the cloud environment variables.

XII. Limits and Quotas

Daily run cap by plan:

Plan	Runs/day
Pro	5
Max	15
Team	25
Enterprise	25

- Orgs with extra usage enabled can exceed the cap on metered overage
- **Minimum schedule interval:** 1 hour
- **Resource limits per run:** 4 vCPUs, 16 GB RAM, 30 GB disk
- **Token budget:** Each run draws from your normal subscription usage. Complex routines burn tokens fast.

XIII. What Persists vs. What Gets Destroyed

Persists:

- claude/ branches pushed to your GitHub repo
- The session on claude.ai (review, continue, create PRs)

Destroyed after each run:

- The cloud environment (cloned repo, temp files, installed packages)

Every run is a clean slate. No state carries over.

XIV. Writing Good Routine Prompts

The prompt IS the routine. A vague prompt gets vague results.

- **Be specific.** "Fix flaky tests" is bad. "Scan src/tests/ for tests that have failed intermittently in the last 5 CI runs, identify the root cause, and open a fix PR for each" is good.

- **Name the files and directories.** Claude is cloning fresh with no prior context beyond CLAUDE.md.
- **State what success looks like.** "All modified tests should pass locally before pushing."
- **Set boundaries.** "Only modify files in src/utils/. Do not touch the API layer."
- **Include the output format.** "Push changes to a claude/fix-flaky-tests branch and open a draft PR with a summary."
- **Leverage CLAUDE.md.** Stable instructions go in CLAUDE.md, run-specific instructions in the prompt. Don't repeat yourself.
- **Tell it what to do on failure.** "If X fails, log the error and stop" vs "if X fails, try Y."

XV. Why This Beats Normal Automation

A typical n8n / Zapier / cron automation runs a fixed sequence. If step 3 fails, the whole thing dies and you get an error log.

Routines are fundamentally different. You're injecting a prompt into a full Claude Code session. You get the **entire agentic reasoning loop**:

- **Self-correction:** If Claude hits an error mid-run, it reads it, reasons about what went wrong, and tries a different approach.
- **Context-aware problem solving:** Claude reads your codebase, checks file contents, looks at test output, adapts. It's not a rigid flowchart.

Nate frames this through his **W.A.T. framework** (Workflows, Agents, Tools): when you push a normal automation to the cloud as a Python script, you lose the **A**gent. Only the workflow and tools survive. With routines, **all three** stay together. The agent reads your CLAUDE.md, looks at your scripts, figures out what to do, and self-corrects mid-run.

Even though each run is stateless, you can configure routines to leave a memory trail (logs, notes back to a doc) so they continuously improve.

XVI. Real-World Use Cases

Event-driven (GitHub triggers):

- PR merge triggers auto-docs update
- PR opened triggers automated code review against CLAUDE.md and style guide
- PRs touching /auth-provider get flagged and summarized to Slack

Scheduled (cron):

- Nightly bug triage: pull top Linear issue at 2am, attempt fix, open draft PR
- Flaky test cleanup
- Weekly backlog grooming
- Dependency audits, code cleanups, release notes on autopilot
- Weekly docs drift detection

API-triggered:

- Fire from CI/CD or Sentry/Grafana alerts, Claude triages and posts to #oncall
- Post-deploy smoke tests: CD pipeline calls routine, Claude runs checks, posts go/no-go

> One user is migrating 8 marketing agents to routines. As Nate puts it: *"It's not just 'AI helps me code faster.' It's 'AI handles entire repeatable workflows autonomously.'"*

XVII. Common Questions

Do I need to know cron syntax? No. Pick presets or use natural language.

Can it access my local files? No. Only what's in your GitHub repo or reachable via APIs.

What models can it use? Any model. Pick when creating the routine.

Can I watch it work in real time? Yes. Hit "Run now" and watch the session live, just like normal Claude Code. You can even talk to it after it's done or interrupt and continue.

Can it use my MCP servers? Not your local ones. It uses Connectors (Anthropic's managed MCP integrations).

Can teammates use my routines? No. They belong to your individual account. Team-plan sharing may be possible but Nate hasn't tested it.

What's the cost? Draws from your normal subscription usage, plus the daily run cap.

What happens when a run fails? Every run creates a session on claude.ai (success or failure). Open it, see what went wrong, retry. No automatic retries. You can also build a fallback into the prompt: *"If this fails, send me a Slack message."*

Can I test before going live? Yes, and you should, multiple times. Hit "Run now," watch it execute, inject corrections, build confidence. Then trust the schedule.

XVIII. Final Setup Checklist

Before trusting any routine on a schedule:

1. Use **"Run now"** and watch the session live
2. Verify all API keys are in the **cloud environment variables** (not .env)
3. Confirm **network access** is set correctly (Trusted vs. Full)
4. Make sure the prompt explicitly tells Claude where to find env vars if needed
5. Make sure the prompt tells Claude what to do when it hits a wall
6. Confirm your CLAUDE.md isn't dragging in irrelevant context
7. Test failure paths, not just the happy path
