Claude Code + Trigger.dev: Build Insane AI Agents

This guide covers how to use Claude Code to build workflows and agents in plain English, then deploy them to Trigger.dev so they run automatically in the cloud, no babysitting required.

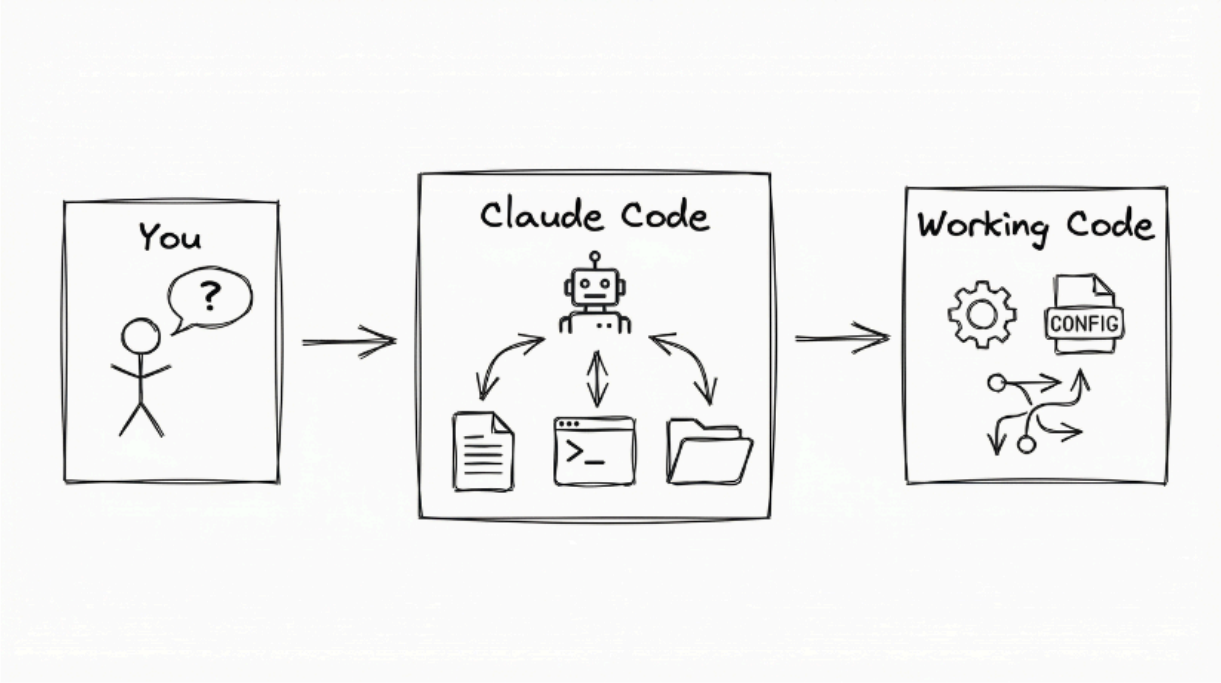
By: Nate Herk

I. The Big Picture: Claude Code + Trigger.dev

The combination works like this:

1. **Claude Code**: You describe what you want in plain English. Claude Code turns your vague request into actual working automation code (TypeScript files).
2. **Trigger.dev**: You push those TypeScript files to Trigger.dev so they can run on a schedule, retry on failure, and operate 24/7 in the cloud without your laptop being open.

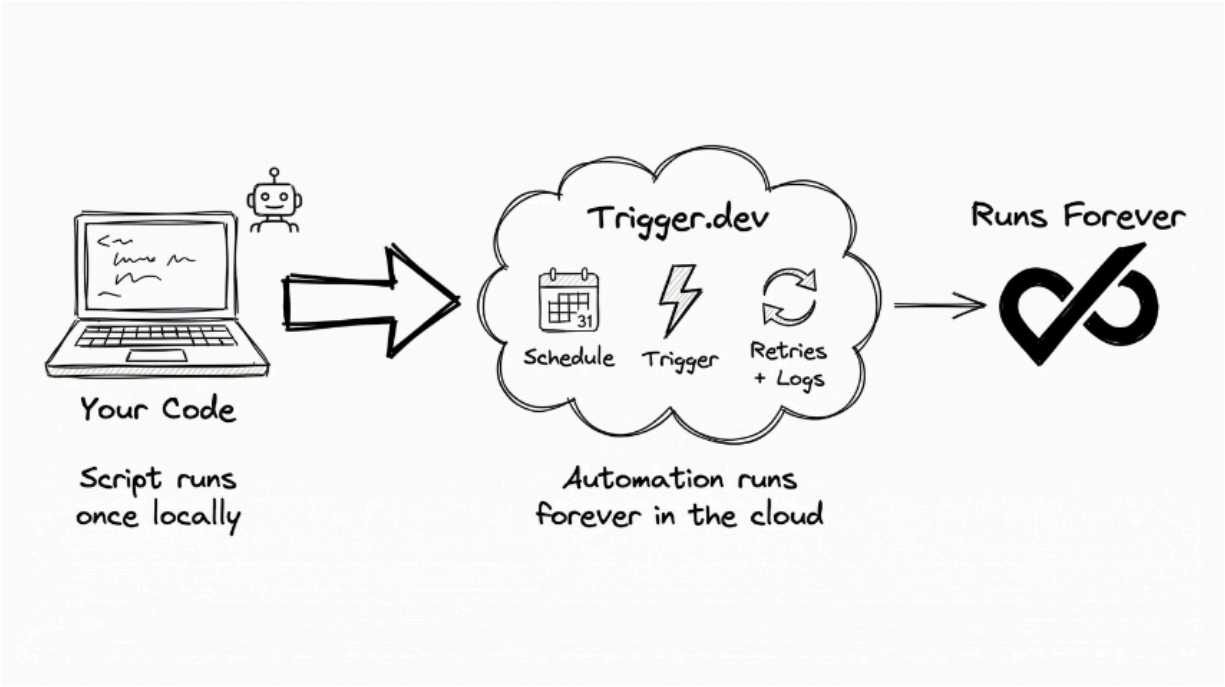
This flow lets you go from idea to live, production-ready automation incredibly fast, even for complex agents with multiple tools and decision-making capabilities.



II. Why Trigger.dev Over Other Options (e.g., Modal)

Trigger.dev offers several advantages for running automations in the cloud:

- **Scheduled runs**: Set automations to fire on a cron schedule (e.g., every Monday at 8 AM, every 2 minutes, every 8 hours).
- **Automatic retries**: If a task fails, Trigger.dev adds a delay and tries again automatically.
- **Queuing & orchestration**: Break complex workflows into separate tasks that can be managed independently. If one piece fails, only that piece retries instead of the whole automation breaking.
- **Live run monitoring**: Watch every step an agent takes in real time, including how long each step takes.
- **Clean UI**: Easy-to-navigate interface with runs, schedules, and environment management.
- **Dev & Production environments**: Test in development, then push to production when ready.



III. What Was Built in the Video (Demo Overview)

Automation 1: AI News Digest A workflow that checks if Nate B. Jones (AI news channel) has posted a new YouTube video. If yes, it generates key highlights, concepts, quotes, and stats. If no, it does nothing. Built in roughly 10 minutes.

Automation 2: Company Research Agent (ClickUp Integration) A non-deterministic agent that watches a ClickUp list. When a new company is added as a task:

- The agent triggers, does web research about the company, and leaves a detailed research brief as a comment on the task.
- The agent can also **conversate**. You can ask follow-up questions (e.g., "Does this company have a recent valuation?") and it does additional research and responds.
- This is a true agent with multiple tools (search web, read URL) that decides which loops to enter and when it has enough information.

Architecture of the ClickUp Agent:

- **3 tools/tasks:** Process Video, Responder Agent, Researcher Agent
- **3 scheduled tasks:** YouTube Checker (every 8 hours), Research Poller (every 2 minutes), Follow-Up Poller (every 2 minutes)

The pollers check ClickUp for new tasks or new comments, then hand off to the appropriate agent.

IV. Project Structure

When you build workflows with Claude Code for Trigger.dev, the file structure looks like this:

```
None
your-project/
├── .env                          ← Your secret API keys (never
pushed to GitHub)
├── CLAUDE.md                     ← Instructions for Claude Code
├── trigger-ref.md                ← Trigger.dev API reference &
TypeScript patterns
└── src/
    └── trigger/
        ├── ai-news-digest/
```

```
│   ├── process-video.ts
│   └── youtube-check.ts
└── company-research/
    ├── research-poller.ts
    ├── follow-up-poller.ts
    ├── researcher-agent.ts
    └── responder-agent.ts
```

Each TypeScript file is a task or agent that gets pushed to Trigger.dev.

V. Step-by-Step: Building Your First Automation

Step 1: Set Up Your Project

Open a new, blank folder in VS Code with Claude Code running. This is your starting point.

Step 2: Grab the Resource Files

Go to the free **Skool community** → Classroom → Claude Code → Trigger.dev section. Download these two files and drag them into your project:

- **`CLAUDE.md`**: Tells Claude Code how to behave, what its goals are, and where to look for code patterns.
- **`trigger-ref.md`**: The Trigger.dev API reference with TypeScript examples. The CLAUDE.md file references this so Claude Code can look up patterns when writing code.

Step 3: Give Claude Code a Prompt

You can start with a vague request. Example from the video:

> "I need you to build me an automation that goes off every single Monday and searches the web to find me dental practices that I can sell websites to."

Claude Code will read the CLAUDE.md file and then ask clarifying questions such as:

- Where should leads be delivered? (e.g., ClickUp)
- What location should it search? (e.g., Nationwide)

- Do you have any API subscriptions set up? (e.g., SERP API, Google Maps)
- How many leads? (e.g., 25 leads per batch)

Step 4: Review the Plan

Claude Code will present an architecture plan before building. For the dental lead example:

- **Task 1: Find Leads** searches for dental practices using SERP API (free alternatives like Yelp Fusion may not work anymore).
- **Task 2: Create Leads in ClickUp** takes each found lead and creates a task in a dedicated ClickUp list.
- **Idempotency built in** ensures deduplication so the same practice never gets added twice.
- **Schedule** is set to run every Monday at 8 AM.

The separation into two tasks is important: if one fails, only that task retries instead of the whole automation breaking.

Step 5: Set Up Your Environment Variables

Claude Code will create a `.env` file with placeholders. You need to fill in your API keys:

```
None
SERP_API_KEY=your_key_here
CLICKUP_API_KEY=your_key_here
TRIGGER_SECRET_KEY=your_key_here
```

Important: Never paste API keys directly into the Claude Code chat. Always tell it to put a placeholder in the `.env` file, then you fill it in manually.

Step 6: Connect to Trigger.dev (Development)

1. Create a new project in Trigger.dev.
2. Go to **Project Settings** and copy the **Project Ref**.
3. Give this to Claude Code so it can connect your local project to Trigger.dev.
4. Your tasks should now appear in the Trigger.dev dev environment.

Step 7: Add Environment Variables to Trigger.dev

Your `.env` keys don't get pushed to Trigger.dev automatically. You need to add them manually:

1. Go to Trigger.dev → **Environment Variables** (at the bottom of the sidebar).

2. Click **Add New**.
3. You can copy your entire `.env` file contents and paste them in. Trigger.dev will parse all the keys at once.
4. **Add to both Development AND Production.** There's no point doing just one since you'll eventually push to production.

Step 8: Optional: Add the Trigger.dev MCP Server

For better testing capabilities, drag in the MCP configuration file (also available in the Skool classroom). This lets Claude Code interact with Trigger.dev more directly for triggering test runs and managing tasks.

VI. Testing & Iteration

- Run tests in the **Development** environment first.
- Watch runs live in Trigger.dev to see every step and debug issues.
- **Plan mode matters.** The video showed that one-shot prompting can lead to issues (like incomplete deduplication logic). Using Claude Code's plan mode and being very clear about requirements upfront saves time on iteration.
- Talk to Claude Code to fix issues. Example: "I did a test run and some leads were duplicates. Can you fix the idempotency logic?"

VII. Pushing to Production

Once your automation works in development, you need to push it to production so it runs without your local dev server being connected.

Option A: GitHub Integration (Recommended)

1. Tell Claude Code to push the project to a GitHub repository.
2. In Trigger.dev → **Project Settings** → **Connect a GitHub Repo**.
3. Select your repo and connect it.
4. Every push to the master branch automatically deploys to the production environment.

Why GitHub?

- Version control
- Cloud backup
- Collaboration with other engineers
- Automatic syncing with Trigger.dev

Security reminder: The `.env` file is automatically excluded from GitHub commits (anything with a dot before it is hidden from commits). Never put secrets directly in your codebase.

Option B: Manual Deploy

If GitHub isn't your thing, you can tell Claude Code to look at the Trigger.dev docs and manually push to the production environment via CLI.

VIII. Key Concepts & Takeaways

Deterministic vs. Non-Deterministic Automations

- **Deterministic:** A simple 1-2-3-4-5 workflow that always follows the same steps.
- **Non-deterministic (Agent):** Has multiple tools, makes decisions about which path to take, and determines when it has gathered enough information. The ClickUp research agent is an example of this.

The Dev to Production Pipeline Build in Claude Code → Test in Trigger.dev Dev environment → Push to GitHub → Auto-deploy to Trigger.dev Production. Development requires your local server to be connected. Production runs independently.

Every Run Is Different Claude Code uses AI models that think and make decisions. Even with the same prompts, you might get slightly different results each time. This is normal. Just talk to it, ask what's wrong, and guide it.

Your Role Has Changed You no longer have to write code, but you are the person who assures quality and keeps the AI on track. The skill is in clearly communicating what you want and verifying the output.

Plan Before You Build The video demonstrated that one-shot prompting can work for simple things, but more complex workflows benefit greatly from using plan mode and having a clear idea of requirements before telling Claude Code to build.
