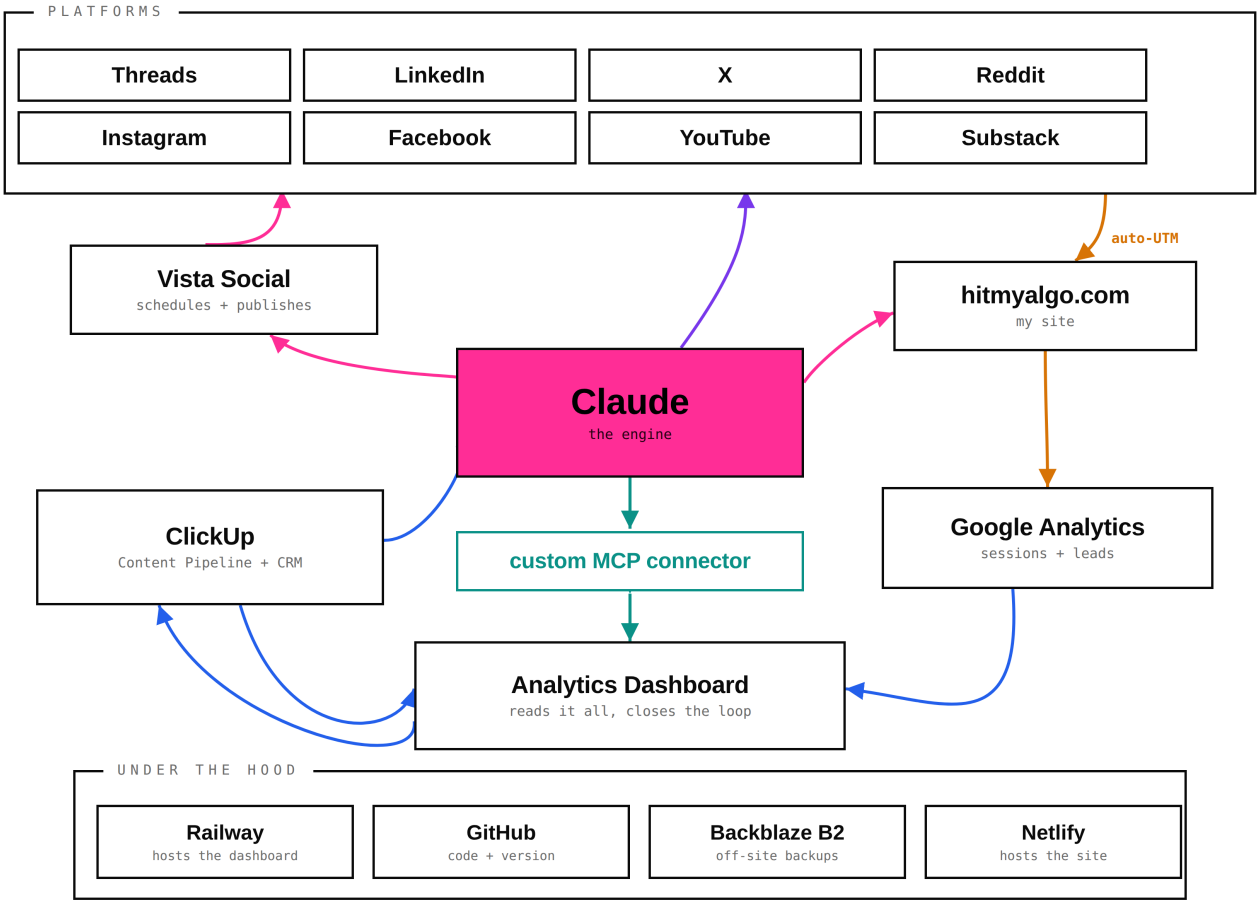
THE SYSTEM

One loop.

Every post traced to a lead.



Ship	Content drafted with Claude, scheduled into Vista, articles to the site.
Engage	Claude drafts the comments and DMs. You read and edit each before it sends.
Traffic	Links auto-UTM-tagged for the life of the post. GA ties each click to it.
Measure	Vista, GA, and ClickUp all feed the dashboard. One source of truth.
Run	Claude runs the dashboard itself through a custom MCP connector you built.

What this is

This is exactly how I run my marketing agency. Content, engagement, outreach, publishing, and measurement, wired into one loop through MCP connectors.

No Zapier. No duct tape. The AI operates the tools directly. The last piece went in just before I wrote this: a custom MCP connector for the analytics dashboard I built. The same chat that drafts and schedules now reads the numbers and keeps the board honest. I can run the entire operation without leaving one conversation. Not as a demo. As the actual workday.

This is not an autoposter. Nothing publishes on its own. Every post, comment, and DM gets my eyes, my review, and my edits before it goes out. The system does the fetching, drafting, tagging, scheduling, and measuring. The judgment stays mine. That never changes anywhere in this guide.

This guide is the blueprint and the checkpoints. Claude is the manual. I'm not going to hand you fifty pages of screenshots showing which button to click. Those would be wrong within months, because these tools move their menus constantly. Every step tells you WHAT to build, gives you the exact words to ask Claude for the HOW, and ends with a check so you know it worked before you move on.

Part 1 shows you the machine and the one idea that makes it work. Part 2 builds it.

THE ONE IDEA

The join key

This is the piece almost nobody does, and it's what makes the whole loop traceable. Every piece of content gets ONE name, used in three places:

- The ClickUp task name: `Threads: measurement lesson`
- The link tag: every owned-domain link in that post gets `utm_campaign=measurement-lesson`
- The Vista post it matches, by platform and publish time

Same name everywhere. That name is the join key.

Now GA4 doesn't just say "someone visited from social." It says WHICH POST sent them, and whether they converted. Post to click to session to lead. One unbroken chain per piece.

Skip the naming discipline and the chain breaks. The dashboard can only join what shares a key.

REAL NUMBERS · MY ACCOUNT

What it caught

The reason to build this is what it shows you. From my own account:

- My most liked post: 11,928 impressions, 187 engagements. Clients from it: **zero.**
- A post with 12 views and no likes: 4 website sessions. **3 leads. One client.**

The platform's analytics would have told me to make more of the first post. The loop told me the second one paid. And one recent week, measured across every channel:

One week: impressions up 493% past 38,000, engagement up over 300%, Threads followers up 21%, likes up 590%.

That week I posted less than the week before. Fewer, better pieces, each traced to what it earned.

Track what pays you, not what flatters you.

WHAT YOU NEED · 1 OF 2

The honest scope

What you're getting into: this build wires several services together through their APIs. ClickUp, GA4, your scheduler, and Backblaze each get connected with keys and endpoints, then everything deploys as one app and gets hooked into Claude. You direct Claude through all of it, so you're not writing code from scratch. But you are doing real integration work: tokens, endpoints, environment variables, a deploy. That's the honest scope.

It's a living system, not a one-time build. Things break. APIs change. You'll find pieces you want to add. Expect to fix a bug now and then and to keep improving it long after the first version runs. Claude is how you do it each time.

How long it takes:

- You've wired up APIs or built with Claude Code before: **a weekend.**
- Technical enough to follow along but this hookup is new to you: **a week of evenings.** Nothing here is hard on its own, there are just several pieces to connect, and you'll check each as you go.
- You've never touched an API or a deploy: this probably isn't the project to cut your teeth on. Read Part 1, skip Part 2, and use it to hold whoever runs your marketing to a real standard.

Not sure which is you? Don't guess.

> ASK CLAUDE
>
> "Is this build beyond my skill level, and how long will it really take ME?"

If you've built anything with Claude before, ask it straight and take the answer seriously. That's more honest than any line I could write here.

WHAT YOU NEED · 2 OF 2

The tools

- **Claude (Max plan)**: $100 to $200/mo depending on your usage tier. Start at $100, move up if you hit limits running this daily. The engine. Drafts everything, operates every tool, writes the code with you, answers every "how do I..."
- **ClickUp**: free. The backbone. Content pipeline + CRM.
- **Vista Social**: ~$150/mo. Publishing. **Hard requirement: the tier that includes the MCP connector.** Confirm the plan lists both an MCP connector and REST API access before you pay. The connector runs the scheduling, the API feeds the dashboard. Without the connector, the run-it-from-chat workflow doesn't exist.
- **Google Analytics 4**: free. Outcomes.
- **Railway (or Render)**: ~$5/mo. Hosts the dashboard you'll build.
- **Backblaze B2**: free to start (first 10GB free). Off-site backups.
- **GitHub**: free. Version control.

Not on the list: wherever your website already lives (mine's on Netlify). You host your site however you already do, GA4 just goes on it.

All in: about $255 to $355/mo depending on your Claude tier, and most of that is Claude and Vista. Turn on 2FA on every new account, and set Railway's usage limit so a bug can't surprise-bill you.

The one rule that carries the whole build: any time you don't know how to do something, or anything errors, or a term means nothing to you, ask Claude right there in the chat. Every step includes the exact words to ask. That's not a crutch. That's the manual.

Before you start: know what a "lead" is for your business. A form submit, a booking click, a purchase. Your conversion is whatever pays you. Every step points at it.

BEFORE YOU START

Before you let an AI touch your machine

Read this before you grant anything. It matters more than any step. An AI coding tool works by getting access to parts of your computer. Access is real: a folder grant means it can read what's in that folder.

- **Work in a dedicated project folder.** Make one folder for this build and grant access to that folder only. The AI needs your project, not your life.
- **Never point it at your personal documents.** Not Documents, not Desktop, not Downloads. A grant you clicked through without reading can expose tax returns, IDs, passwords.
- **Read every permission prompt.** That's a decision, not a formality. If you don't understand it, ask the AI to explain the permission before you grant it.
- **Handle your keys yourself.** YOU paste each key directly into the place it belongs, the service's dashboard and Railway's environment variables. You don't hand raw keys to the AI in chat. Keys have two homes: a password manager while you hold them, Railway environment variables once the app uses them. Never a loose file, a public repo, a post, or a screenshot. If a key leaks, don't agonize, rotate it, about a minute per service.
- **Gate every write.** Any action that changes something real requires a preview and an explicit confirm.

Same reason you don't give a new contractor the keys to every room. Scope the access, and you've closed the biggest risk in the whole build.

BEFORE YOU START

When you get stuck

You will get stuck. Everyone does. A step won't work, an error will make no sense, a menu won't match. That is not the build failing. That is the build. Here's how to get unstuck fast instead of rage-quitting:

- **Take a screenshot and hand it to Claude.** Show it exactly what you're looking at. It reads screenshots.
- **Tell it three things:** what you were trying to do, where you are right now, and why you're confused. "I'm on the Railway environment variables screen, adding my ClickUp token, and I don't know which field is the name and which is the value." That gets a real answer. "It's broken" does not.
- **Paste the real error.** The full text, not your summary. The error is the clue.
- **Work with it, not at it.** Nothing comes out perfect first try, including for me. Expect a few rounds on the harder steps. That's the normal shape, not a sign you're doing it wrong.

One safety rule before you send a screenshot: make sure no API keys, passwords, or private info are visible. Crop or black them out first.

Every problem in here has an answer, and the fastest way to it is a clear screenshot and an honest description of where you're stuck.

STEP 1

Set up the board

What you're building: the system's memory. One ClickUp list where every piece of content is a task and every status is the truth.

You do this once:

1. Create a free ClickUp account and one list called `Content Pipeline`.
2. Give it exactly four statuses: **Draft, Approved, Scheduled, Published.**
3. Connect ClickUp to Claude: add the ClickUp connector in Claude's settings, so the chat can create and move tasks for you.

That's your whole job here. Two minutes.

The AI does everything else: it creates every task, names every task, and moves every task through the statuses. You'll almost never touch this board by hand.

The standing order you give the AI (the naming rule, the entire trick): every content task is named `Platform: short title`, like `Threads: measurement lesson`. That short title is the slug, the join key from Part 1.

Put this standing order where every new chat can see it: a Claude Project's instructions, not a single conversation. Same for the tagging rule in Step 3.

> ASK CLAUDE
>
> `"Walk me through creating a list in ClickUp with custom statuses, step by step."`

> **You'll know it worked when:**
>
> you tell Claude "log a test piece called hello world for Threads" and a task named `Threads: hello world` appears in your list as Draft.

STEP 2

Connect the scheduler

What you're building: one publishing point, so the dashboard only has to look one place to know what went live.

You do this once:

1. Create a Vista Social account on the MCP tier and connect every profile you post to.
2. Paste Vista's MCP connector URL into Claude's connector settings. This is the piece that lets Claude schedule for you.
3. Tell Claude to schedule one test post.

The AI does from then on: all scheduling, from chat, at the times you set, and later the times your data says work. After it schedules, it lists the posts back so you see each one landed. You still approve every post before it's scheduled. Nothing goes out that you didn't read.

> ASK CLAUDE
>
> `"Where do I add an MCP connector URL in Claude's settings? Walk me through it."`

> **You'll know it worked when:**
>
> you ask Claude "list my scheduled posts" and the test post comes back in the answer.

STEP 3

Tag every link

What you're building: receipts. A UTM tag is just extra text on the end of a link (`yoursite.com/?utm_campaign=measurement-lesson`) that tells GA4 exactly where a click came from. Untagged links all blur into "social." Tagged links testify.

You never write these by hand. I don't. I couldn't write one from memory and I run this system every day. The AI builds the tag into every link at scheduling time.

> THE STANDING ORDER YOU GIVE THE AI, ONCE
>
> "Every time you schedule a post, tag every link to a domain I own with utm_source = the platform, utm_medium = social, and utm_campaign = the task's slug. Never schedule an owned-domain link untagged."

You do: nothing per-post. That's the point.

> **You'll know it worked when:**
>
> you click a link from one of your own scheduled posts and watch the session show up in GA4's realtime view with the campaign name attached.

STEP 4

GA4, the money signal

What GA4 is for: sessions are traffic, key events are the business. GA4 is where a click becomes a session and a session becomes a recorded lead. The dashboard reads from it. The UTM tags from Step 3 are what let it tell one post from another.

Fair warning: this is the step that fights you. GA4's settings are a maze, the menus move, and it will test your patience. Mine tested mine. This is exactly what the ask-Claude rule and the stuck page are for.

If GA4 isn't on your site yet: create a property and install the tag before anything else, then come back. Ask Claude to walk you through it.

Set up your key events (once):

1. Mark each action that pays you as a key event, one at a time. The exact words are below.
2. Fire each one yourself once to confirm it registers.
3. Note it: you just put fake conversions in your own data. Remember that.

> ASK CLAUDE
>
> `"Walk me through marking a form submit as a key event in GA4, step by step, current UI."`

Two things about GA4 that will bite you: GA4 isn't realtime for reporting, the standard tier can lag hours, so treat the dashboard as end-of-day truth, not a stock ticker. And you'll pollute your own data, the only fake leads my system ever logged were me testing my own forms. Label or filter your test sessions.

> **You'll know it worked when:**
>
> your test action shows up as a key event within a day, sometimes two, and a tagged link shows in realtime with its campaign name.

STEP 5 · 1 OF 2

Build the dashboard

What you're building: the small app that joins everything into one view. ClickUp knows the plan, Vista knows the posts, GA4 knows the outcomes. Nothing knows all three until you join them on the slug. You are the foreman here, not the bricklayer.

Set your expectations first, this is where people quit: you are not dropping in one prompt and getting a finished dashboard back. That prompt is the START of a conversation. Claude writes a version, you run it, something errors, you paste the error back, it fixes it, you go again. That loop is the work. Plan for several passes, not one.

A rule that saves you on the big moves: before you let Claude make a large change, ask it to write the plan first and check its own plan before building.

> ASK CLAUDE, BEFORE ANY BIG CHANGE
>
> "Before you make this change, write out your plan, then run a subagent to review that plan for problems and report back before you touch any code."

Catching a bad plan on paper is cheap. Catching it after it's built is not. The exact build spec is on the next page.

STEP 5 · 2 OF 2

Build the dashboard

You do this once:

1. Gather your credentials: a ClickUp API token, a GA4 service account (create it in Google Cloud, enable the Analytics Data API, grant its email Viewer on your GA4 property, download the JSON key), your GA4 numeric property ID, and a Vista Social REST API key. Keep them in your password manager for now.
2. Open Claude Code in your dedicated project folder and paste the spec below. Build and test it locally first, a .env file listed in .gitignore is the one acceptable home for keys while you test.
3. When something doesn't work, that's the stuck page: screenshot, describe, paste the error, keep moving.
4. Deploy on Railway: create a project, attach a volume for the SQLite trend store, connect the private GitHub repo Claude made, then re-enter each key as a Railway environment variable (base64 the GA4 JSON key). Railway builds it and gives you a URL.

PASTE THIS SPEC INTO CLAUDE CODE

```
"Build me a marketing dashboard as a Node + Express app with a single-file frontend. Three data sources: the ClickUp API, the Vista Social REST API, and the GA4 Data API. Read every credential from environment variables, never from a file in the repo. Join all three on a slug: the ClickUp task title after the FIRST colon, lowercased with spaces turned to hyphens and punctuation stripped, must match the utm_campaign parsed from each post's own tagged link (fallback: match by platform plus publish time within 30 minutes, all normalized to UTC). One view per content piece showing status, impressions, engagement, sessions, and leads. Persist a daily snapshot of the joined metrics to SQLite on a Railway volume, that's the trend history. Put a simple password or access token in front of the whole dashboard. Include a Dockerfile for Railway. Walk me through every step, assume I've never deployed an app, and pause at each step so I can confirm."
```

You'll know it worked when:

you open your Railway URL and see one piece of content with its post metrics and its sessions on the same row.

STEP 6

Wrap it in an MCP connector

What you're building: the loop-closer. This turns your dashboard from a thing you look at into a thing you talk to. Every other connector in this system belongs to someone else's product. This one you own.

Same as Step 5: an iterative conversation, not a one-shot prompt. For a change this central, ask Claude to write and review its plan before building.

You do this once: back to Claude Code, paste the spec, redeploy, then paste the server URL into Claude's connector settings, same as Vista's in Step 2.

PASTE THIS SPEC INTO CLAUDE CODE

```
"Add an MCP server to the dashboard, HTTP transport, on the same Railway deployment. Require a secret token on every request, a long random token in the URL path stored in a Railway environment variable, since claude.ai connectors can't send custom headers; I'll paste that same token as part of the connector URL. Read tools for: summary, performance, attribution, best content, pipeline state, trends. Write actions for reconcile and send-digest, but stub send-digest until the alert channel exists in Step 8. Every write must require an explicit confirm flag and a matching dry-run preview tool. Before you build, write your plan and run a subagent to review it, then show me the plan before you touch code."
```

You'll know it worked when:

you ask Claude "show me my content summary for the last 7 days" and your own numbers come back in the chat. Then run the reconcile preview and see exactly what it would change, without it changing anything.

STEP 7

Back it up

What you're building: protection for the one thing you can't re-download. Every API here can be re-queried except your own trend history. Lose it and your trendlines start over.

You do this once:

1. Create a Backblaze B2 account and bucket. Enter its keys into Railway's environment variables yourself. You hand Claude the variable names, not the raw keys.
2. Have Claude add the nightly backup job with the spec below.
3. Redeploy.

> ASK CLAUDE
>
> ```
> "Add a nightly job that copies the trend snapshot from the SQLite volume to
> Backblaze B2 over the S3-compatible API, keep the newest 14 copies, and write a
> restore script that validates the file and writes atomically."
> ```

> **You'll know it worked when:**
>
> you restore yesterday's snapshot into a scratch folder and it opens clean.

STEP 8

Set the watchdogs

What you're building: the parts that keep the system honest while you're not looking. Not autonomy. Safety nets. They watch and they alert. They don't post.

You do this once, in the same Claude Code project.

First, pick one alert channel (an email service API key or a Slack webhook) and put its key in Railway's environment variables. The watchdogs need somewhere to send.

> ASK CLAUDE
>
> "Add a weekly watchdog that audits every scheduled piece against what published and flags mismatches. Add a daily check that catches posts I made natively from my phone: read my published posts from the Vista REST API and flag any that aren't in the pipeline (if my Vista tier only exposes scheduled posts and not published ones, skip this one). Add a health monitor that runs every few hours, compares against the stored daily snapshots, and alerts me on my chosen channel if a data source stops responding or if impressions drop more than 60% week over week."

Then redeploy. Without these, a failed schedule or a phone post posted off-pipeline quietly falls out of your numbers. The watchdogs are what let you trust the board without babysitting it.

> **You'll know it worked when:**
>
> you post something natively from your phone, and by the next day it shows up in your dashboard as an untracked piece instead of vanishing.

STEP 9

Run the loop

What you're building: nothing. The build is done. This is what operating it looks like, and it's the part that compounds.

The daily loop:

1. Draft in chat. The AI logs the piece to ClickUp as Draft, named by the rule.
2. You approve. Nothing unapproved ever ships. You read and edit every piece.
3. The AI schedules it into Vista at the right slots, tags the links, lists it back to confirm it landed, moves it to Scheduled.
4. Once it's live, reconcile advances it to Published with metrics attached. Preview first, confirm, done.
5. The watchdogs run in the background.

Notice your job in that list: draft, approve, confirm, edit. Judgment. The machine does the fetching and the filing. It is not an autoposter and it never becomes one.

> **You'll know it worked when:**
>
> you wake up, ask one question in chat, and get back what published, what it earned, and what to make next.

KNOW THESE

Things that will bite you

(GA4's two gotchas live in the GA4 section. Here's everything else.)

- **Some media won't attach through APIs.** The Vista connector can't attach locally generated images, so image posts cost thirty seconds on the platform. The text pipeline needs no platform time.
- **Long text into web forms can garble.** Long text typed into a web form can mangle, and platforms like LinkedIn expect a human at the keyboard anyway. Draft in chat, paste it yourself.
- **Gate every write action.** Preview plus explicit confirm, no exceptions. The first time an AI edits your board for real, you want to have seen the exact list of changes before you say yes.
- **This isn't autonomy, and it isn't an autoposter.** Every DM, comment, and post gets human eyes, a review, and edits before it ships. The system replaces the guessing and the busywork, not the judgment.
- **It's a living system.** You'll hit bugs. Tools will change their APIs. You'll think of things to add. Fixing and improving it over time is part of owning it, and Claude is how you do each fix.

THE PAYOFF

What you built

You now have:

- A board kept honest, checked weekly against what really published.
- Links that testify which post sent the click.
- A dashboard that joins plan, post, and payoff on one key.
- A connector that lets you run all of it by talking.
- Watchdogs that keep it honest while you sleep.

One loop. Every lead traced back to its post.

On your accounts, your site, your dashboard. Walk away from any agency and you keep the entire thing. That's the point.

Questions? I give this stuff away. DM me and I'll answer.

hitmyalgo.com · @SpragueSocials · adam@hitmyalgo.com