How to build your own AI agent operating system from scratch.

A complete build manual. No coding background required. Paste-ready prompts throughout.

This is the full architecture for a personal AI agent operating system. Not just a chatbot. An actual operating system that knows your goals, manages your calendar, captures your thoughts, and coaches you toward what matters.

The architecture is split into four phases. Each phase builds on the last. You can stop after Phase 1 and have a working assistant. Each subsequent phase makes it more powerful.

Follow it in order. Don't skip phases. The dependencies matter.

Built with Claude Code | Node.js | Telegram | Google Calendar | Obsidian | Railway

THE ARCHITECTURE

Four phases. Build in order.

Each phase adds a capability layer. You can stop at any phase and have a useful agent. But the magic happens when all four are stacked.

PHASE 1

Foundation

Get a Telegram bot live with your custom personality. Connect it to your calendar. Deploy to the cloud so it runs 24/7.

PHASE 2

Input Layer

Add voice notes, screenshots, and document uploads. Build the capture system so getting thoughts INTO the agent is frictionless.

PHASE 3

Intelligence Layer

Connect a structured knowledge base (Obsidian). Add note classification so captured items auto-route to the right place.

PHASE 4

Proactivity Layer

Calendar actions (create, update, delete events). Smart scheduling that fills free time with goal-relevant tasks. Morning briefs.

WHAT YOU'LL END UP WITH

Your own AI agent that:

- Lives on Telegram. Reachable from anywhere.
- Knows your goals, voice, and operating principles.
- Reads your calendar across multiple accounts.
- Accepts voice notes, screenshots, and text input.

- Captures everything to a structured knowledge base.
- Routes notes automatically (tasks, ideas, contacts, etc).
- Manages your calendar (create, move, delete events).
- Proactively surfaces priorities and flags drift.
- Costs roughly $10-20/month to run all-in.

PHASE 1

Foundation

Get a working Telegram bot live with calendar awareness and your personality baked in. End of Phase 1 = usable assistant.

STEP 01

Pick your foundation tools

Before building anything, get clear on the stack. Each tool has a free tier or low entry cost.

- **Claude Code** — the AI builder. Writes the code for you. Comes with a Claude Pro subscription.
- **GitHub** — stores your code. Free for personal use.
- **Railway** — runs your agent online 24/7. Around $5/month.
- **Telegram** — where your agent talks to you. Free.
- **Google Calendar** — what the agent reads to manage time. Free.
- **Obsidian** — knowledge base (added in Phase 3). Free.
- **Anthropic API** — intelligence layer. Pay-as-you-go, typically under $10/month.

Sign up as you go. Save every login in a password manager.

STEP 02

Get your API keys

API keys are how the agent connects to each service. Treat them like passwords.

Get these three before starting:

- **Anthropic API key** — at console.anthropic.com, create new. Starts with *sk-ant-*
- **Telegram bot token** — message @BotFather on Telegram, send /newbot, follow prompts. You'll get a token like *1234567890:ABC...*
- **Telegram chat ID** — message your new bot once, then visit *https://api.telegram.org/bot<TOKEN>/getUpdates* in a browser. Find the chat.id number in the response.

Critical: never share these keys publicly. If anyone gets the Anthropic key, they can spend your account credit.

STEP 03

Set up your project folder

This is where the agent code lives on your machine.

Create a clean folder. Open Terminal (Mac) or Command Prompt (Windows):

```
mkdir ~/my-ai-agent
cd ~/my-ai-agent
```

On Mac, install GitHub CLI so Claude Code can push your work online:

```
brew install gh
gh auth login
```

When prompted: choose **GitHub.com**, then **HTTPS**, then **login with browser**. Follow the flow.

STEP 04

Build the bot with Claude Code (personality first)

This is the most important step in the entire build. Define your agent's personality BEFORE writing code. The voice, the rules, the defaults. This is what separates an AI agent from a generic chatbot.

Start Claude Code from inside your project folder:

```
claude --dangerously-skip-permissions
```

Paste the prompt below. Customise the personality section to fit you:

> *"I want to build a personal AI agent that runs my life through Telegram. Before we write any code, define WHO this agent is. Then we build around that.*
>
> ***The agent's personality** (this is what makes it mine, not generic):*
> *• Role: My personal executive assistant and chief of staff*
> *• Voice: Direct, warm, no corporate filler, no AI sentence-starters (Let me, Great question, I'd be happy to)*
> *• Communication style: Short. Bullet points when scanning helps. No em dashes, no semicolons. Recommend ONE path, not options.*
> *• Default behaviour: Proactive. Surfaces priorities. Calls me out when I'm drifting.*
> *• Always knows: my goals, schedule, non-negotiables, voice.*
>
> ***Phase 1 scope**:*
> *1. Receive Telegram messages*
> *2. Reply using Claude with the custom persona*
> *3. Log every conversation*
> *4. Be deployable to Railway for 24/7 running*
>
> ***Build instructions:***
> *1. Node.js project with CommonJS modules*

2. Folder structure: /api for webhooks, /lib for helpers, /scripts for local dev
3. Build the Telegram webhook handler
4. Build a Claude API wrapper with the persona loaded as system prompt
5. Initialise git, create a private GitHub repo, push it
6. Write a README explaining how to deploy to Railway
7. Set up .env.local with placeholders for keys

Walk me through each step. Wait for confirmation before moving forward. Use the voice rules in the persona system prompt."

Claude Code walks you through file by file. Approve each step.

Customise the personality before pasting. The example uses one voice. Yours should match how you actually want to be spoken to.

STEP 05

Wire up Telegram

Add the keys from Step 2 to your **.env.local** file:

```
ANTHROPIC_API_KEY=sk-ant-your-key-here
TELEGRAM_BOT_TOKEN=your-bot-token
TELEGRAM_CHAT_ID=your-chat-id
```

Test locally with *npm start*. The bot should boot. Send it a Telegram message; nothing happens yet because Telegram doesn't know where your local server is. That gets fixed when you deploy in Step 8.

STEP 06

Connect Google Calendar

Tell the agent to read your calendar.

OAuth setup:

- Go to **console.cloud.google.com**, create a new project
- Enable the Google Calendar API
- Create OAuth 2.0 credentials (Web application type)
- Set redirect URI to **https://developers.google.com/oauthplayground**
- Save the Client ID and Client Secret

Generate a refresh token at developers.google.com/oauthplayground:

- Click the settings gear, paste your Client ID and Secret, check 'Use your own OAuth credentials'

- Add scopes: *calendar.readonly* and *calendar.events*
- Authorise with your Google account, then exchange code for tokens
- Copy the refresh_token

Add to .env.local:

```
GOOGLE_CLIENT_ID=your-client-id
GOOGLE_CLIENT_SECRET=your-client-secret
GOOGLE_REFRESH_TOKEN=your-refresh-token
```

Now tell Claude Code:

> *"Add Google Calendar integration. Read all calendars the connected account has access to (including shared calendars). Default to fetching the next 7 days. Support natural language date queries: today, tomorrow, this week, next week, next month, specific weekdays. Use my timezone (set GOOGLE_CALENDAR_TIMEZONE env var). When listing events, tag each with a category emoji based on the source calendar (e.g. work, personal, partnerships). Update the bot so when I ask what is on my calendar, it pulls events and replies grouped by day."*

Pro move: share other Google calendars (work, personal) TO your main one. The agent sees all of them with one OAuth grant.

STEP 07

Set the timezone and week conventions

Tell the agent where you are and how your week works. Otherwise it defaults to UTC and Sunday-start weeks, which breaks reasoning about 'this week' and 'tomorrow'.

In Claude Code:

> *"Update the calendar code to use [your timezone, e.g. Pacific/Auckland] for all date calculations. Set week to start Monday and end Sunday. Make sure parseDateRange functions use these conventions everywhere. Update the persona system prompt to mention the timezone and week structure so the agent always replies in local time."*

Small but critical. Without it, ambiguous queries like 'what is tomorrow' return wrong-day results.

STEP 08

Deploy to Railway

The agent only works while your laptop runs. To go 24/7, deploy to Railway.

- Go to **railway.app**, create a new project, deploy from your GitHub repo
- Once deployed, generate a public domain in **Networking** settings
- Add every variable from your .env.local to Railway's **Variables** tab

- Update the Telegram webhook so messages route to Railway:

```
curl https://api.telegram.org/bot<TOKEN>/setWebhook?
url=https://your-railway-url.up.railway.app/api/telegram-webhook
```

Send a Telegram message. The bot replies from the cloud. Phase 1 complete.

Phase 1 milestone reached. You now have a working personal AI agent on Telegram. Stop here and use it for a week before adding more. The agent gets smarter as you use it and discover what you actually need.

PHASE 2

Input Layer

Make capture frictionless. Voice notes, screenshots, document uploads. Whatever's in your head should reach the agent in seconds.

STEP 09

Add voice note transcription

Voice is faster than typing, especially for messy thoughts. Add OpenAI's Whisper model to transcribe voice notes sent via Telegram.

Setup:

- Sign up at **platform.openai.com**
- Generate an API key (starts with *sk-*)
- Add $5 credit. Whisper costs $0.006 per minute, so $5 covers ~14 hours of audio

Add to .env.local and Railway:

```
OPENAI_API_KEY=sk-your-key-here
```

Tell Claude Code:

> *"Add voice note transcription via OpenAI Whisper. When Telegram sends a voice message, download the audio file, send it to Whisper for transcription, then process the transcribed text the same way a typed message would be processed. Confirm receipt to the user with the transcription so they can verify accuracy. If transcription contains ambiguous names or technical terms, ask for clarification before acting."*

Voice messages now feel native. You think out loud, the agent captures it.

STEP 10

Add image and screenshot understanding

Claude can read images. Send a screenshot of an Instagram profile, an article, or a meeting note, and Claude can extract what's in it.

Tell Claude Code:

> *"Add image input handling. When Telegram sends a photo, download it, send to Claude with vision enabled, and ask Claude to describe what is in the image. Then process the description as context for further conversation. For social profile screenshots, extract:*

> *handle, bio, follower count, topics they post about. For article screenshots: extract the headline, key claims, source if visible. Confirm what was extracted before doing anything with it."*

Anything you can screenshot becomes structured input the agent can act on.

STEP 11

Add document upload support

PDFs, Word docs, and other files all become readable input.

Tell Claude Code:

> *"Add document upload handling. When Telegram sends a document (PDF, DOCX, TXT, MD), download it, extract the text content, and send it to Claude as context. For PDFs, use a PDF parser library. Ask the user what they want to do with the document: summarise, extract action items, save to knowledge base, etc."*

Send the agent a meeting transcript, a contract, a research paper. It can summarise, extract actions, or file it away.

Phase 2 milestone reached. You can now feed the agent anything: text, voice, images, documents. Capture friction is solved. Phase 3 is where it gets *smart* about what you feed it.

PHASE 3

Intelligence Layer

Connect a structured knowledge base. Teach the agent who you are, what you want, what matters. Then make it route input automatically.

STEP 12

Design your knowledge base structure

Obsidian becomes the agent's long-term memory. Your goals, voice, businesses, contacts, daily logs all live here as markdown files.

Recommended folder structure:

- **00_BRAIN/** — rarely changes. Profile, Vision, Goals, Operating Principles, Voice and Brand
- **01_COMMAND/** — weekly state. This Week, Open Loops, Pipeline
- **02_AREAS/** — one folder per business venture or major project
- **03_AGENTS/** — definitions for each AI agent (so you can scale to multiple)
- **04_LOGS/** — daily logs, weekly reviews, agent output history
- **05_INBOX/** — unprocessed: voice notes, ideas, screenshots, links
- **06_MEMORY/** — long-term recall (what worked, what didn't, patterns)
- **07_KNOWLEDGE/** — reference material, frameworks, books
- **08_RELATIONSHIPS/** — your CRM: sales pipeline, partnerships, etc

Numbered prefixes sort the folders in workflow order automatically.

STEP 13

Push your vault to GitHub

Production agents can't see your laptop, so the vault needs to live on GitHub for the agent to read it.

- Create a private repo on GitHub (e.g. *my-agent-vault*)
- Open Terminal in your Obsidian vault folder
- Run: *git init && git add . && git commit -m 'initial' && git push*

Generate a Personal Access Token with read+write access to that single repo:

- Go to *github.com/settings/tokens?type=beta*
- Create fine-grained token, repository access: only this vault repo

- Permissions: Contents → Read and write
- Save the token securely

Add to .env.local and Railway:

```
OBSIDIAN_GITHUB_TOKEN=your-token
OBSIDIAN_GITHUB_OWNER=your-username
OBSIDIAN_GITHUB_REPO=your-vault-repo
OBSIDIAN_GITHUB_BRANCH=main
```

STEP 14

Wire the agent to read the vault

Tell the agent which files to read on every message.

In Claude Code:

> *"Add Obsidian integration. Read specific markdown files from a GitHub repo using a personal access token, and pass them as system context to Claude on every message. Maintain two file lists: a lightweight list (BRAIN files + current week state) loaded on every message, and a full list for deeper analysis. Skip files that contain only placeholder content. Cache reads for 5 minutes to reduce API calls. Format the loaded context cleanly as system context."*

The agent now grounds every reply in your actual goals and voice. Generic answers become personalised answers.

STEP 15

Fill the BRAIN with real content

This is the step most people skip and then wonder why their agent feels generic.

Inside **00_BRAIN/**, write real content into each file. Not aspirational. Real:

- **Profile.md** — who you are, how you work, your rhythm, what to assume by default
- **Vision.md** — where you're going long-term. For you to re-read on bad days
- **Yearly_Goals.md** — specific numbers. North Star metrics. Sprint targets
- **Quarterly_Goals.md** — what this 90 days looks like
- **Businesses.md** — every venture, what each does, current state
- **Operating_Principles.md** — non-negotiables. Schedule, focus rules, money rules
- **Voice_and_Brand.md** — how you sound. Words you use. Words you never use

Each file should be a few hundred words minimum. The richer the content, the better the agent.

Push to GitHub after each file. The agent picks up changes within minutes.

STEP 16

Add note classification and auto-routing

Now that input is captured (Phase 2) and a structured knowledge base exists (steps 12-15), connect them. Voice notes, ideas, screenshots get classified and routed automatically.

Tell Claude Code:

> *"Add note classification. When the user sends a voice note, screenshot, or text capture, classify it as one of: TASK, IDEA, GOAL_UPDATE, REFLECTION, CONTACT, REFERENCE. Then route accordingly:*
> *• TASK → append to 01_COMMAND/Open_Loops.md with today's date and a checkbox*
> *• IDEA → create a new file in 05_INBOX/Ideas/[timestamp]-[slug].md*
> *• GOAL_UPDATE → notify user and ask if they want to update 00_BRAIN/Yearly_Goals.md*
> *• REFLECTION → append to today's 04_LOGS/Daily/[date].md*
> *• CONTACT → ask which pipeline (Sales/Podcast/Partnership) and create stub file in 08_RELATIONSHIPS*
> *• REFERENCE → save to 07_KNOWLEDGE with extracted title*
>
> *For each classification, commit the change to GitHub. Confirm to the user what was filed where."*

This is when the agent becomes genuinely useful. Voice-note a podcast guest idea while walking and find it filed in your CRM when you sit back down.

Phase 3 milestone reached. The agent now has memory, context, and the ability to organise input automatically. Phase 4 makes it proactive.

PHASE 4

Proactivity Layer

The agent acts on its own. Creates events. Suggests focus. Sends morning briefs. Coaches you toward your goals without being asked.

STEP 17

Add calendar write actions

The agent reads your calendar. Now let it write.

Tell Claude Code:

> *"Add calendar write actions: createEvent, updateEvent, deleteEvent. All write actions must show a confirmation summary to the user (title, time, duration, calendar) and wait for explicit YES before executing. Support natural language: book 1 hour Tuesday at 10am for content writing. Default to the user's primary calendar unless they specify. Include a description with the source of the event (e.g. created from voice note on 2026-05-12)."*

The agent can now manage your time, not just observe it. Always with confirmation, never autonomously.

STEP 18

Build the smart scheduling engine

This is the agent at full power. It looks at your calendar, your open loops, your goals, and your operating principles, then suggests how to spend free time.

Tell Claude Code:

> *"Build a smart scheduling function. When triggered (manually via what should I do or proactively in the morning brief):*
> *1. Read the next 7 days of calendar*
> *2. Identify gaps of 30+ minutes within working hours (defined in Operating_Principles.md)*
> *3. Read Open_Loops.md, This_Week.md, and Yearly_Goals.md*
> *4. Rank pending tasks by their fit to North Star goals*
> *5. Suggest which task belongs in which gap, with reasoning that references the user's goals*
> *6. Offer to create calendar events for accepted suggestions*
>
> *Always frame suggestions as recommendations, never autonomous bookings. Show reasoning so the user can override."*

Now you ask what should I do and the answer is grounded in your actual goals, not generic productivity advice.

STEP 19

Add the morning brief

A scheduled job that sends a brief to Telegram each morning. Pulls today's calendar, top open loop, top goal-relevant task, and (optionally) AI news.

On Railway: add a cron schedule (Settings → Cron). Run a brief generator at your wake-up time.

Tell Claude Code:

> *"Build a morning brief generator. Each weekday at [your wake time, e.g. 6:45am], send a Telegram message containing:*
> *1. Today's date and weather (optional, requires weather API)*
> *2. Today's calendar events grouped by category emoji*
> *3. The ONE highest-priority open loop, with context*
> *4. The ONE focus block suggestion (largest gap of the day, plus what to put in it)*
> *5. Three sentences of latest AI news (optional, requires news API like Tavily)*
>
> *Keep the whole brief under 200 words. Scannable in under 90 seconds. End with one direct call to action: what to do first."*

You wake up, glance at Telegram, know exactly what matters today. No decision fatigue.

STEP 20

Add proactive nudges

The final piece. The agent flags when you're drifting from what you said you'd do.

Tell Claude Code:

> *"Add a daily drift-check cron job. Each evening, read 04_LOGS/Daily/[today].md and compare against This_Week.md priorities. If today's logged work doesn't align with the week's top three priorities, send a Telegram message flagging the drift. Use a direct, coaching tone. Cite specifics: you said priority X this week, today's logged work was Y. Ask what happened. If logged work DOES align, send a brief encouragement noting the alignment."*

The agent now closes the accountability loop. Goals → daily work → reflection → adjustment.

Phase 4 milestone reached. The agent is now a full operating system. It captures, organises, plans, executes (with permission), and holds you accountable. Maintenance from here is mostly content updates to the BRAIN files as your business evolves.

YOU'RE DONE

You've built a full AI operating system.

Four phases. Twenty steps. A working assistant that knows your goals, captures your thoughts, organises your work, and coaches you toward what matters.

The hard part wasn't the code. The hard part was being honest about what you want this agent to do for you. That clarity is what makes it yours.

From here, it's iteration. Add new capabilities as you discover what you actually need. The architecture supports it. Each new feature is another conversation with Claude Code, another commit, another deploy.

DON'T WANT TO BUILD IT YOURSELF?

Have it built for you.

AI Consultancy builds custom AI agents and operating systems for founders and businesses that want the result without the build. From single-task agents to full personalised operating systems like the one in this guide.

aiconsultancy.co.nz

Book a free 30-minute discovery call to scope what your AI agent could do.

Built with Claude | aiconsultancy.co.nz