Personal AI Agent Setup Guide

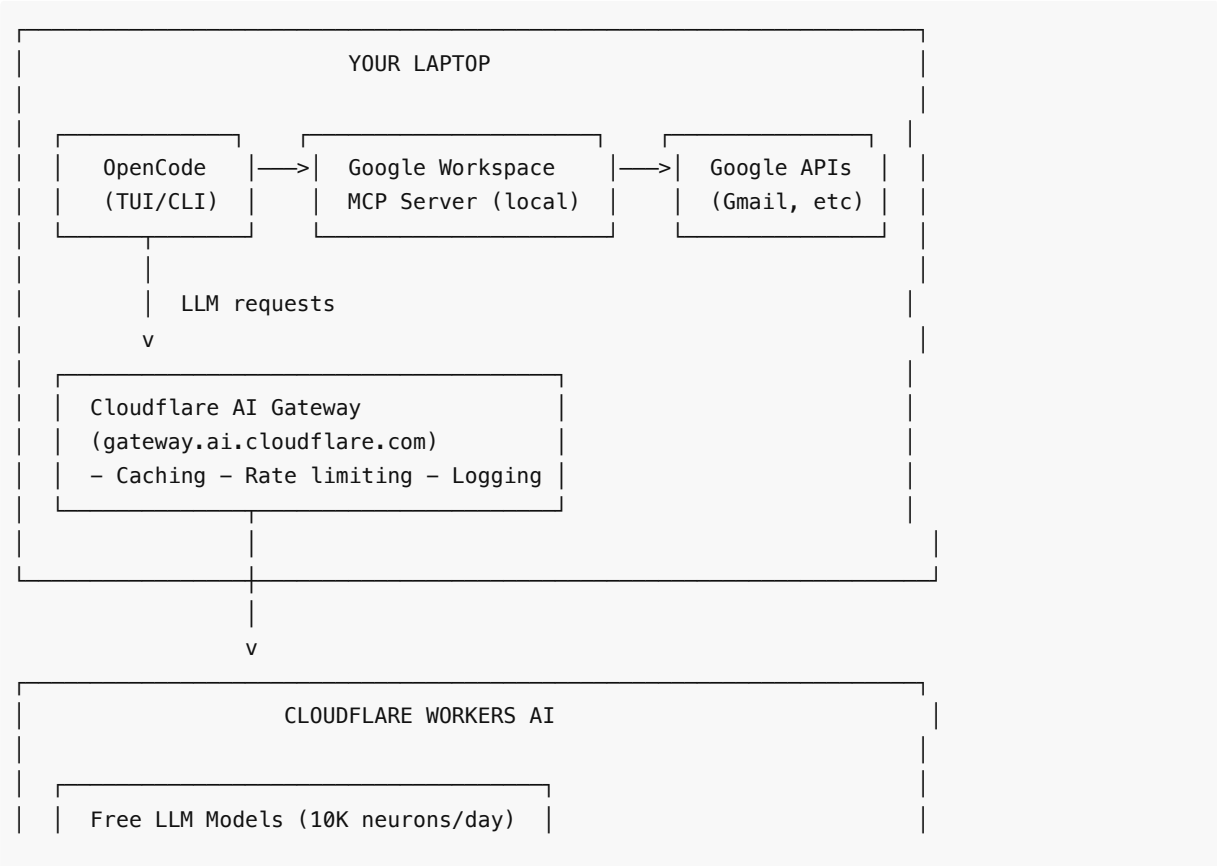
Secure AI with Cloudflare Workers, MCP Servers, AI Gateway & OpenCode

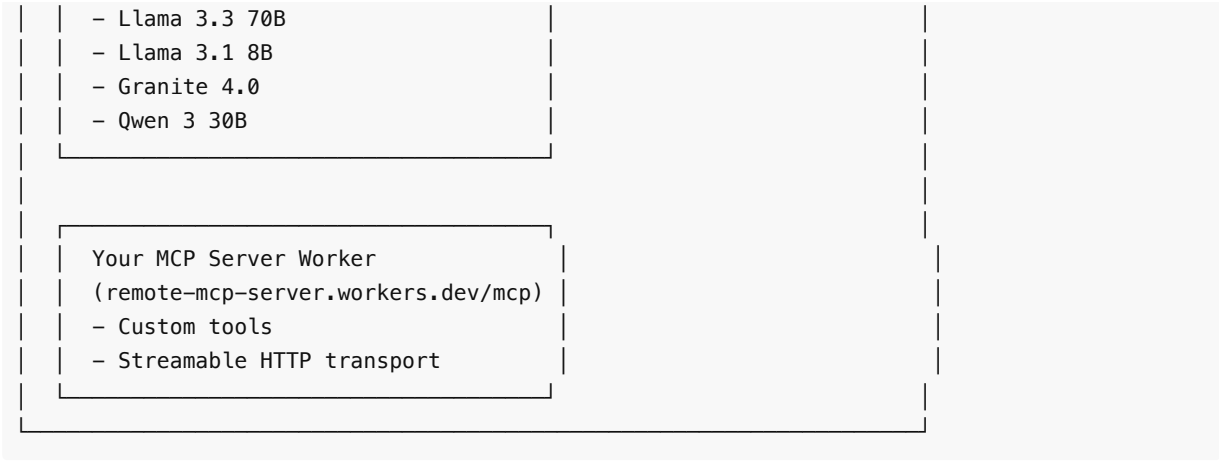
For: Sean Connellan (sean.connellan@gmail.com) **Cloudflare Plan:** Free **Goal:** A personal AI assistant that can read your email, manage your calendar, edit your documents, search your drive, and run custom skills — all powered by free Cloudflare LLM models, secured behind AI Gateway, and controlled from your terminal.

Table of Contents

---

1. Architecture Overview





Data flow:

1. You type a prompt in OpenCode
2. OpenCode sends the LLM request through AI Gateway to Cloudflare Workers AI
3. Workers AI processes the request and returns a response
4. If the AI decides to use a tool (e.g., read your email), OpenCode calls the Google Workspace MCP server running locally on your machine
5. The MCP server uses your OAuth credentials to call Google APIs
6. Results flow back through the AI for further processing

---

2. What Each Component Does

OpenCode (The Client)

What it is: A terminal-based AI coding assistant — think of it as your personal AI command centre that runs in your terminal.

What it does:

- Provides a text-based interface (TUI) for chatting with AI models
- Manages connections to LLM providers (Cloudflare Workers AI in our case)
- Connects to MCP servers to give the AI access to external tools
- Loads "skills" — reusable instruction sets that teach the AI how to perform specialised tasks
- Handles file editing, code search, and shell command execution
- Manages permissions (what the AI is allowed to do on your machine)

Why you need it: Without a client like OpenCode, you would have to manually construct API calls to the LLM and manually invoke MCP tools. OpenCode orchestrates all of this automatically.

MCP (Model Context Protocol) Server

What it is: A standardised protocol that lets AI models discover and invoke external tools. An MCP server exposes a set of "tools" that an AI client can call.

What it does:

- Advertises available tools (e.g., "I can search Gmail", "I can create calendar events")
- Receives tool invocation requests from the AI client
- Executes the tool (calling the appropriate API)
- Returns structured results back to the AI

Two types you will use:

Type	Location	Purpose

Local MCP server	Runs on your laptop	Google Workspace access (Gmail, Calendar, Drive, Docs, etc.) — needs your OAuth credentials
Remote MCP server	Runs on Cloudflare Workers	Custom tools you deploy to the cloud — accessible from anywhere

Why you need it: Without MCP, the AI cannot interact with any external system. MCP is the bridge between the AI and the real world.

AI Gateway

What it is: A Cloudflare service that sits between your AI client and the LLM provider, acting as a proxy for all AI requests.

What it does:

- **Caching** — Stores identical requests so repeat queries are served instantly (saving neurons/tokens)
- **Rate limiting** — Prevents runaway AI usage from burning through your free allocation
- **Logging** — Records every request/response for debugging and auditing
- **Cost control** — Tracks spending across all your AI calls
- **Model routing** — Can switch between models based on rules you define
- **DLP (Data Loss Prevention)** — Can redact sensitive data before it reaches the LLM

Why you need it: On the free plan, you get 10,000 neurons per day. AI Gateway's caching alone can dramatically extend this by serving cached responses instead of making new LLM calls. It also gives you visibility into what the AI is doing.

Cloudflare Workers AI (The LLM)

What it is: Cloudflare's serverless GPU infrastructure that runs open-source LLM models at the edge.

What it does:

- Runs inference on LLM models hosted on Cloudflare's global network
- Charges by "neurons" (a unit of compute) rather than per-token
- Provides an OpenAI-compatible API endpoint

Free models available (selected highlights):

Model	Size	Neurons/M input tokens	Best for
`@cf/ibm-granite/granite-4.0-h-micro`	Small	1,542	Quick tasks, classification
`@cf/meta/llama-3.2-1b-instruct`	1B	2,457	Fast, simple queries
`@cf/meta/llama-3.1-8b-instruct-fp8-fast`	8B	4,095	General purpose (recommended)
`@cf/qwen/qwen3-30b-a3b-fp8`	30B	4,577	Reasoning tasks
`@cf/meta/llama-3.3-70b-instruct-fp8-fast`	70B	26,668	Complex tasks (uses neurons fast)

Why you need it: This is the "brain" of your AI assistant. On the free plan, you get 10,000 neurons/day — enough for ~2.4M input tokens with the 8B model, or ~375K input tokens with the 70B model.

Cloudflare Workers (The MCP Server Host)

What it is: A serverless compute platform that runs your code on Cloudflare's edge network.

What it does:

- Hosts your remote MCP server as a Worker
- Provides a URL (e.g., `your-mcp.workers.dev/mcp`) that any MCP client can connect to
- Handles HTTP requests, routing, and scaling automatically
- Can optionally use Durable Objects for stateful sessions

Why you need it: If you want your MCP server accessible from anywhere (not just your laptop), you need to deploy it on Workers. This is also useful for sharing tools with others.

Google Workspace MCP Server

What it is: A local MCP server (running on your laptop) that connects to Google's APIs and exposes your Gmail, Calendar, Drive, Docs, Sheets, Slides, and Chat as tools the AI can use.

What it does:

- Authenticates with your Google account via OAuth
- Exposes 50+ tools: read/send email, create/edit docs, manage calendar events, search Drive, read/write spreadsheets, manage slides, send Chat messages, etc.
- Runs as a local process — your Google credentials never leave your machine
- Handles OAuth token refresh automatically

Why local, not remote? Running it locally means:

- Your OAuth credentials and tokens stay on your machine
- No need to store secrets in Cloudflare environment variables
- Lower latency for interactive use
- Simpler setup — no need to deploy or manage a remote server

Skills

What they are: Markdown files (`SKILL.md`) that contain specialised instructions for the AI. They teach the AI how to perform specific tasks.

What they do:

- Inject domain-specific knowledge into the AI's context when invoked
- Provide step-by-step workflows the AI should follow
- Reference scripts, templates, and other resources
- Can be project-specific or global (available in every session)

Example skills:

- `email-summariser` — Summarises unread emails into a prioritised digest
- `daily-briefing` — Compiles a morning briefing with calendar, emails, and tasks
- `call-notes-summary` — Reformats messy call notes into structured summaries

Why you need them: Skills turn a general-purpose AI into a specialist. Without skills, the AI would need to be re-taught the same workflows every session.

Profile

What it is: A Markdown file (`profile.md`) that tells the AI who you are and what context it should keep in mind.

What it does:

- Gives the AI persistent context about your identity, role, and preferences
- Prevents you from having to repeat who you are in every conversation
- Can include timezone, communication preferences, and account information

---

3. Prerequisites

Before starting, ensure you have:

- [ ] A macOS or Linux laptop (Windows with WSL also works)
- [ ] Node.js v20+ installed (`node --version` to check)
- [ ] npm installed (`npm --version` to check)
- [ ] A Google account (sean.connellan@gmail.com)
- [ ] A web browser for OAuth flows
- [ ] A terminal / command line

Install Node.js if needed:

```
# macOS (Homebrew)
brew install node

# Or via nvm (recommended)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
nvm install --lts
```

---

4. Step 1 — Create a Cloudflare Account

If you don't already have a Cloudflare account:

1. Go to https://dash.cloudflare.com/sign-up
2. Sign up with your email (sean.connellan@gmail.com)
3. Verify your email address
4. You are now on the **Free plan** — no credit card required

Find your Account ID

1. Go to https://dash.cloudflare.com
2. Click any domain (or "Workers & Pages" in the sidebar)
3. Your Account ID is shown in the right sidebar under "API"
4. Copy it — you will need it later

Create an API Token

1. Go to https://dash.cloudflare.com/profile/api-tokens
2. Click **Create Token**
3. Use the **"Edit Cloudflare Workers"** template
4. Under Permissions, add:
   - Account — Workers Scripts — Edit
   - Account — Workers AI — Edit
   - Account — Account Settings — Read
5. Click **Continue to summary** → **Create Token**
6. **Copy the token now** — it will not be shown again

Save these as environment variables:

```
# Add to ~/.zshrc or ~/.bashrc
export CLOUDFLARE_ACCOUNT_ID="your-32-character-account-id"
export CLOUDFLARE_API_KEY="your-api-token"
```

Then reload your shell:

```
source ~/.zshrc  # or source ~/.bashrc
```

---

5. Step 2 — Set Up AI Gateway

AI Gateway sits between OpenCode and Workers AI, providing caching, logging, and rate limiting.

Create a Gateway

1. Go to https://dash.cloudflare.com
2. Navigate to **AI** → **AI Gateway** in the sidebar
3. Click **Create Gateway**
4. Name it: `personal-ai-gateway`
5. Click **Create**

Your gateway URL will be:

```
https://gateway.ai.cloudflare.com/v1/{ACCOUNT_ID}/personal-ai-gateway/workers-ai/v1
```

Replace `{ACCOUNT_ID}` with your actual Account ID.

Configure Gateway Settings

In the AI Gateway dashboard, you can enable:

Setting	Recommendation	Why
Caching	Enabled	Saves neurons by serving cached responses for identical prompts
Rate Limiting	Enabled, 50 req/min	Prevents runaway usage
Logging	Enabled	See what your AI is doing
DLP	Optional	Redacts sensitive data (emails, phone numbers) before sending to the LLM

What AI Gateway Gives You on the Free Plan

Feature	Free Tier
Core features (analytics, caching, rate limiting)	Included
Number of gateways	10 per account
Log storage	100,000 total across all gateways
Log ingestion rate	500 logs/sec/gateway
Cache TTL	Up to 1 month
DLP	2 predefined profiles
Logpush	Not available (paid only)

---

6. Step 3 — Deploy an MCP Server on Workers

This step creates a **remote MCP server** — a Worker that exposes tools over the internet. This is **optional** if you only want the Google Workspace MCP server (which runs locally). But if you want custom cloud-hosted tools, follow

along.

Create the MCP Server

```
# Create a new MCP server from the official template
npm create cloudflare@latest -- sean-mcp-server --template=cloudflare/ai/demos/remote-mcp-authless

cd sean-mcp-server
```

During setup:

- "Do you want to add an AGENTS.md file?" → **No**
- "Do you want to use git for version control?" → **No**
- "Do you want to deploy your application?" → **No**

Understand the Code

Open `src/index.ts`. It will look something like this:

```
import { createMcpHandler } from "agents/mcp";
import { McpServer } from "@modelcontextprotocol/sdk/server/mcp.js";
import { z } from "zod";

function createServer() {
  const server = new McpServer({ name: "My MCP Server", version: "1.0.0" });

  // Each server.tool() call defines a tool the AI can invoke
  server.tool(
    "hello",                          // Tool name
    { description: "Returns a greeting message", inputSchema: { name:
z.string().optional() } },
    async ({ name }) => ({
      content: [{ text: `Hello, ${name ?? "World"}!`, type: "text" }],
    }),
  );

  return server;
}

export default {
  fetch: (request: Request, env: Env, ctx: ExecutionContext) => {
    const server = createServer();
    return createMcpHandler(server)(request, env, ctx);
  },
} satisfies ExportedHandler<Env>;
```

Key concepts:

- `McpServer` — Creates an MCP server instance that holds your tool definitions
- `server.tool()` — Registers a tool with a name, description, input schema (using Zod), and handler function
- `createMcpHandler()` — Wraps the MCP server into a Worker fetch handler
- The `/mcp` path is where MCP clients connect

Add Your Own Tools

Edit `src/index.ts` to add custom tools. For example:

```
server.tool(
  "get_weather",
  { description: "Get current weather for a city", inputSchema: { city: z.string() } },
  async ({ city }) => {
    const response = await fetch(
      `https://api.openweathermap.org/data/2.5/weather?
q=${city}&appid=${env.WEATHER_API_KEY}`
    );
    const data = await response.json();
    return {
      content: [{ type: "text", text: `Weather in ${city}: ${data.weather[0].description},
${Math.round(data.main.temp - 273.15)}C` }],
    };
  },
);
```

Test Locally

```
# Start the development server
npm start
# -> Ready on http://localhost:8788

# In another terminal, test with MCP Inspector
npx @modelcontextprotocol/inspector@latest
# -> Opens at http://localhost:5173
# -> Enter URL: http://localhost:8788/mcp
# -> Click Connect -> List Tools
```

Deploy

```
npx wrangler@latest deploy
```

Your MCP server is now live at:

```
https://sean-mcp-server.<your-subdomain>.workers.dev/mcp
```

(Optional) Add Authentication with Cloudflare Access

If you want to protect your MCP server so only you can use it:

1. Go to **Cloudflare Zero Trust** dashboard
2. Navigate to **Access** → **Applications**
3. Click **Add an application** → **Self-hosted**
4. Set the application domain to your Worker URL
5. Create a policy that allows only your email (sean.connellan@gmail.com)
6. Follow the guide at: https://developers.cloudflare.com/cloudflare-one/access-controls/ai-controls/secure-mcp-servers/

---

7. Step 4 — Set Up Google Cloud OAuth Credentials

The Google Workspace MCP server needs OAuth credentials to access your Google account. This is the most involved step, but you only do it once.

Create a Google Cloud Project

1. Go to https://console.cloud.google.com/
2. Click the project dropdown (top left) → **NEW PROJECT**
3. Name it: `MCP Workspace Server` → **CREATE**
4. Select the new project from the dropdown

Enable Required APIs

Go to **APIs & Services** → **Library** and enable each of these:

API	What it enables
Gmail API	Reading, sending, searching email
Google Calendar API	Viewing and creating events
Google Drive API	Searching and managing files
Google Docs API	Reading and editing documents
Google Sheets API	Reading and editing spreadsheets
Google Slides API	Reading and editing presentations
Google Chat API	Reading and sending chat messages
People API	Contact information
Tasks API	Google Tasks

You can enable them via the console, or use gcloud CLI:

```
gcloud services enable \
  gmail.googleapis.com \
  calendar-json.googleapis.com \
  drive.googleapis.com \
  docs.googleapis.com \
  sheets.googleapis.com \
  slides.googleapis.com \
  chat.googleapis.com \
  people.googleapis.com \
  tasks.googleapis.com
```

Configure the OAuth Consent Screen

1. Go to **APIs & Services** → **OAuth consent screen**
2. Choose **External** (required for personal @gmail.com accounts — "Internal" only works for Google Workspace org accounts)
3. Fill in:
   - **App name:** `MCP Workspace Server`
   - **User support email:** sean.connellan@gmail.com
   - **Developer contact:** sean.connellan@gmail.com
4. Click **Save and Continue**
5. On the **Scopes** page, click **Add or Remove Scopes** and add:

```
https://www.googleapis.com/auth/gmail.modify
https://www.googleapis.com/auth/calendar
https://www.googleapis.com/auth/drive
https://www.googleapis.com/auth/documents
https://www.googleapis.com/auth/spreadsheets
https://www.googleapis.com/auth/presentations
https://www.googleapis.com/auth/chat.spaces
https://www.googleapis.com/auth/chat.messages
https://www.googleapis.com/auth/contacts
https://www.googleapis.com/auth/tasks
```

> ***Note:*** *Using* `.modify` *instead of* `.readonly` *for Gmail allows sending. If you only want read access, use* `gmail.readonly` *instead.*

6. Click **Save and Continue**
7. On the **Test Users** page, add sean.connellan@gmail.com
8. Click **Save and Continue**

> ***Important:*** *While the app is in "Testing" mode, only test users can authorise. This is fine for personal use. You don't need to submit for Google verification.*

Create OAuth Credentials

1. Go to **APIs & Services** → **Credentials**
2. Click **+ CREATE CREDENTIALS** → **OAuth client ID**
3. Application type: **Desktop Application**
4. Name: `MCP Workspace Client` → **CREATE**
5. **Download the JSON file** — this contains your Client ID and Client Secret
6. Save it somewhere secure:

```
mkdir -p ~/.google-mcp
mv ~/Downloads/client_secret_*.json ~/.google-mcp/credentials.json
```

> ***Security note:*** *This file contains secrets. Never commit it to git or share it. The* `.google-mcp` *directory should have restricted permissions.*

---

8. Step 5 — Install & Configure the Google Workspace MCP Server

This is the MCP server that gives your AI access to Gmail, Calendar, Drive, Docs, Sheets, Slides, and Chat.

Install the Server

```
# Clone the official Google Workspace MCP server
git clone https://github.com/gemini-cli-extensions/workspace.git ~/projects/google-
workspace-mcp

cd ~/projects/google-workspace-mcp
npm install
npm run build
```

First-Time Authentication

The server needs to authenticate with your Google account. On first use, it will open a browser window for OAuth.

```
# Test that the server starts
cd ~/projects/google-workspace-mcp
node workspace-server/dist/index.js --debug
```

The first time you run this, it will:

1. Open your browser to the Google OAuth consent screen
2. Ask you to sign in with sean.connellan@gmail.com
3. Show a "This app isn't verified" warning — click **Advanced** → **Go to app**
4. Ask you to grant the requested permissions — click **Allow**
5. Save the OAuth tokens locally for future use

After the first auth, the server will start and you can Ctrl+C to stop it. The credentials are now cached.

Auth Management Commands

```
cd ~/projects/google-workspace-mcp

# Check auth status
node scripts/auth-utils.js status

# Force re-authentication (if tokens expire)
node scripts/auth-utils.js clear

# Expire token for refresh testing
node scripts/auth-utils.js expire
```

Headless / Remote Authentication

If you are setting this up over SSH or on a machine without a browser:

```
npm run auth-utils -- login
```

This prints an OAuth URL you can open on any device (phone, another computer). After signing in, paste the credentials JSON back into the CLI.

Update Workflow

When you want to update the server to the latest version:

```
cd ~/projects/google-workspace-mcp
git pull
npm install
npm run build
```

---

9. Step 6 — Install OpenCode

OpenCode is the terminal client that ties everything together.

Install

```
# macOS (Homebrew — recommended)
brew install anomalyco/tap/opencode
```

```
# Or via curl
curl -fsSL https://opencode.ai/install | bash

# Or via npm
npm i -g opencode-ai@latest
```

Verify Installation

```
opencode --version
```

First Launch

```
cd ~
opencode
```

This launches the OpenCode TUI (Terminal User Interface). On first run, it will guide you through connecting an LLM provider.

---

10. Step 7 — Configure OpenCode

OpenCode's configuration lives at `~/.config/opencode/opencode.json` (or `opencode.jsonc` for comments). Create or edit this file with all your settings.

Full Configuration File

Create `~/.config/opencode/opencode.jsonc`:

```
{
  "$schema": "https://opencode.ai/config.json",

  // --- LLM Provider: Cloudflare Workers AI via AI Gateway ---
  //
  // This routes all AI requests through your AI Gateway, which adds
  // caching, rate limiting, and logging before forwarding to Workers AI.
  //
  "model": "cf-gateway/@cf/meta/llama-3.3-70b-instruct-fp8-fast",
  "small_model": "cf-gateway/@cf/meta/llama-3.1-8b-instruct-fp8-fast",

  "provider": {
    "cf-gateway": {
      "npm": "@ai-sdk/openai-compatible",
      "name": "Cloudflare AI Gateway",
      "options": {
        // Replace {ACCOUNT_ID} with your Cloudflare Account ID
        // Replace {GATEWAY_ID} with your gateway name (e.g., personal-ai-gateway)
        "baseURL":
"https://gateway.ai.cloudflare.com/v1/{ACCOUNT_ID}/{GATEWAY_ID}/workers-ai/v1",
        "apiKey": "{env:CLOUDFLARE_API_KEY}"
      },
      "models": {
        "@cf/meta/llama-3.3-70b-instruct-fp8-fast": {
          "name": "Llama 3.3 70B (Fast)"
```

```
      },
      "@cf/meta/llama-3.1-8b-instruct-fp8-fast": {
        "name": "Llama 3.1 8B (Fast)"
      },
      "@cf/qwen/qwen3-30b-a3b-fp8": {
        "name": "Qwen 3 30B"
      },
      "@cf/ibm-granite/granite-4.0-h-micro": {
        "name": "Granite 4.0 Micro"
      },
      "@cf/google/gemma-4-26b-a4b-it": {
        "name": "Gemma 4 26B"
      },
      "@cf/mistralai/mistral-small-3.1-24b-instruct": {
        "name": "Mistral Small 3.1 24B"
      }
    }
  }
},

// --- MCP Servers ---
//
// These give the AI access to external tools.
//
"mcp": {
  // Google Workspace MCP (runs locally on your laptop)
  // Provides: Gmail, Calendar, Drive, Docs, Sheets, Slides, Chat
  "google-workspace-local": {
    "type": "local",
    "command": [
      "sh",
      "-c",
      "cd /Users/sean/projects/google-workspace-mcp && node scripts/start.js"
    ]
  },

  // Your remote MCP server on Workers (optional)
  // Provides: Custom cloud-hosted tools
  "sean-mcp-server": {
    "type": "remote",
    "url": "https://sean-mcp-server.<your-subdomain>.workers.dev/mcp",
    "enabled": true
  }
},

// --- Permissions ---
//
// Controls what the AI is allowed to do on your machine.
// "allow" = automatic, "ask" = requires your confirmation, "deny" = blocked
//
"permission": {
  "edit": "allow",       // Edit files
  "webfetch": "allow",   // Fetch web content
  "bash": {
    "*": "allow",              // Allow all commands by default
    "rm *": "ask",             // Ask before deleting files
```

```
      "rm -rf ~*": "deny",       // Never delete home directory
      "rm -rf /*": "deny",       // Never delete root
      "sudo *": "ask",           // Ask before sudo commands
      "git push*": "deny",       // Don't push without asking
      "git commit *": "ask"      // Ask before committing
    }
  },

  // --- Sharing ---
  "share": "disabled"
}
```

Set Environment Variables

Add these to `~/.zshrc` (or `~/.bashrc`):

```
# Cloudflare credentials
export CLOUDFLARE_ACCOUNT_ID="your-32-character-account-id"
export CLOUDFLARE_API_KEY="your-api-token"
```

Then reload:

```
source ~/.zshrc
```

Create a Profile

OpenCode can load a personal profile that gives the AI context about who you are. Create `~/.config/opencode/profile.md`:

```
# Sean Connellan — Profile

## Identity
- **Name**: Sean Connellan
- **Email**: sean.connellan@gmail.com

## Preferences
- Timezone: [your timezone, e.g., Australia/Perth]
- Communication style: Concise, direct

## Google Account
- Primary email: sean.connellan@gmail.com
- Use this account for all Google Workspace operations
```

Verify Configuration

```
# Launch OpenCode and check configuration
opencode debug config

# List connected MCP servers
opencode mcp list
```

---

11. Step 8 — Create Skills

Skills are reusable instruction sets that teach the AI how to perform specialised tasks. They live as `SKILL.md` files in `~/.config/opencode/skills/<skill-name>/`.

Skill File Format

Each skill is a Markdown file with YAML frontmatter:

```
---
name: my-skill
description: What this skill does (1-1024 chars)
---

## What I do
- Specific behaviour description

## When to use me
Instructions for when the agent should invoke this skill.

## Workflow
1. Step one
2. Step two
3. Step three
```

Name rules: lowercase alphanumeric + single hyphens, 1-64 chars, must match the directory name.

Example: Email Summariser Skill

Create `~/.config/opencode/skills/email-summariser/SKILL.md`:

```
---
name: email-summariser
description: Summarises unread emails from the past 24 hours into a prioritised digest
---

## What I do
I fetch unread emails from the past 24 hours and create a concise, prioritised summary.

## When to use me
When the user asks for an email summary, inbox review, or morning briefing.

## Workflow
1. Use `google-workspace-local_gmail_search` with query `is:unread newer_than:1d`
2. For each email, use `google-workspace-local_gmail_get` to fetch full content
3. Categorise each email as: URGENT, ACTION REQUIRED, FYI, or NEWSLETTER
4. Present a summary in this format:

### URGENT
- **From:** sender — Subject line
  - One-line summary

### ACTION REQUIRED
- **From:** sender — Subject line
  - One-line summary + what action is needed

### FYI
- Bullet list of subjects
```

```
### NEWSLETTERS
- Bullet list of senders
```

Example: Daily Briefing Skill

Create `~/.config/opencode/skills/daily-briefing/SKILL.md`:

```
---
name: daily-briefing
description: Generates a morning briefing with calendar, emails, and tasks
---

## What I do
I compile a daily briefing covering calendar events, priority emails, and any outstanding
tasks.

## When to use me
When the user asks for a "morning briefing", "daily briefing", or "what's my day looking
like".

## Workflow
1. Use `google-workspace-local_calendar_listEvents` to get today's events
2. Use `google-workspace-local_gmail_search` with `is:unread newer_than:1d` for emails
3. Use `google-workspace-local_time_getCurrentTime` for current time context
4. Compile into a structured briefing:

## Daily Briefing — [Date]

### Calendar
- [Time] Event name (attendees count)

### Priority Emails
- URGENT: [subject] from [sender]
- Action needed: [subject] from [sender]

### Summary
- X meetings today, Y unread emails, Z urgent items
```

Discover More Skills

Skills can be shared via Git repositories. To add skills from a repo:

```
cd ~/.config/opencode/skills
git clone <skill-repo-url> <skill-name>
```

---

12. Step 9 — Test Everything

Test 1: AI Gateway + Workers AI

Launch OpenCode and ask a simple question:

```
opencode
> What is the capital of Australia?
```

You should get a response from the LLM. Check your AI Gateway dashboard at https://dash.cloudflare.com → AI → AI Gateway → `personal-ai-gateway` to see the logged request.

Test 2: Google Workspace MCP

```
> What's on my calendar today?
```

The AI should invoke `google-workspace-local_calendar_listEvents` and return your events.

Test 3: Gmail

```
> How many unread emails do I have?
```

The AI should invoke `google-workspace-local_gmail_search` with `is:unread`.

Test 4: Google Drive

```
> Search my Drive for any documents about "project plan"
```

Test 5: Skills

```
> Give me my daily briefing
```

This should invoke the `daily-briefing` skill, which orchestrates multiple MCP tools.

Test 6: Remote MCP Server (if deployed)

```
> Use the hello tool on my MCP server
```

Test 7: Switch Models

In the OpenCode TUI, use `/models` to switch between models. Try the 8B model for fast responses and the 70B model for complex reasoning.

---

13. Do You Need an MCP Portal?

What is an MCP Portal?

An MCP Portal is a centralised service that acts as a single entry point for multiple MCP servers. Instead of connecting to each MCP server individually, the AI client connects to the portal, which then routes tool calls to the appropriate backend server.

Portal features include:

- **Centralised discovery** — One URL to find all available tools
- **DLP (Data Loss Prevention)** — Scans tool inputs/outputs for sensitive data
- **Audit logging** — Records every tool invocation
- **Code Mode** — Collapses N tools into 2 (`portal_codemode_search` + `portal_codemode_execute`), reducing token usage by ~94%
- **Shadow MCP detection** — Detects unauthorised MCP servers via DLP regex patterns

Do You Need One?

For a personal setup with one user: No.

An MCP Portal is designed for enterprise scenarios where:

- Multiple users need access to the same MCP servers
- You need centralised governance and audit logging
- You need to prevent shadow MCP usage
- You need to reduce token usage across many tools

For a personal setup with 1-3 MCP servers, the overhead of setting up and maintaining a portal isn't worth it. You can connect to each MCP server directly in your OpenCode config.

If you want to experiment with it anyway, Cloudflare has published a reference architecture at https://blog.cloudflare.com/enterprise-mcp/ — but this requires a paid Workers plan and is overkill for personal use.

When Would You Add One?

Consider adding a portal if:

- You deploy 5+ MCP servers and want a single URL for all of them
- You share your MCP servers with other people
- You need audit logging for compliance
- Token usage from tool definitions becomes a problem

---

14. Free Plan Limits & Optimisation

What You Get for Free

Service	Free Tier
Workers AI	10,000 neurons/day
Workers	100,000 requests/day
AI Gateway	Core features, 10 gateways, 100K logs
Workers (MCP server)	100,000 requests/day, 100 workers

How Far Does 10,000 Neurons/Day Go?

Model	Neurons/M input tokens	Approx. daily free capacity
Granite 4.0 Micro	1,542	~6.5M input tokens
Llama 3.2 1B	2,457	~4M input tokens
Llama 3.1 8B (Fast)	4,095	~2.4M input tokens
Qwen 3 30B	4,577	~2.2M input tokens
Llama 3.3 70B (Fast)	26,668	~375K input tokens

For context, a typical conversation turn uses ~500-2,000 input tokens (including system prompt, conversation history, and tool results). So with the 8B model, you can have roughly 1,200-4,800 turns per day.

Tips to Maximise Free Usage

1. **Use AI Gateway caching** — Identical prompts are served from cache, consuming zero neurons. Enable it in the gateway settings.
2. **Use the right model for the task** — Use `llama-3.1-8b-instruct-fp8-fast` for simple tasks and `llama-3.3-70b-instruct-fp8-fast` only for complex reasoning.

3. **Set `small_model`** — OpenCode uses the small model for lightweight operations (title generation, etc.), saving neurons:

   ```
   "small_model": "cf-gateway/@cf/meta/llama-3.1-8b-instruct-fp8-fast"
   ```

4. **Keep conversations short** — Start new conversations rather than continuing long ones. Context length grows with each turn.
5. **Monitor usage** — Check your AI Gateway dashboard daily to understand consumption patterns.
6. **Use MCP tool results efficiently** — When the AI calls a tool, the result becomes part of the context. Ask for concise summaries rather than full data dumps.

If You Hit the Limit

The Workers Paid plan ($5/month) gives you:

- Unlimited neurons beyond the 10K/day free allocation (at $0.011/1,000 neurons)
- Longer CPU time (30M CPU-ms/month vs 10ms/invocation)
- More Workers (500 vs 100)
- Logpush for exporting logs

At $0.011/1,000 neurons, 100K additional neurons costs about $1.10 — very affordable for personal use.

---

15. Troubleshooting

OpenCode Can't Connect to Workers AI

Symptom: "Failed to generate" or timeout errors.

Fix:

1. Verify environment variables are set: `echo $CLOUDFLARE_ACCOUNT_ID` and `echo $CLOUDFLARE_API_KEY`
2. Check the API token has Workers AI permissions
3. Verify the AI Gateway URL is correct in your config
4. Check the AI Gateway dashboard for error logs

Google Workspace MCP Server Won't Start

Symptom: "MCP server google-workspace-local failed to start"

Fix:

1. Check Node.js is installed: `node --version`
2. Verify the path in your config matches your actual path
3. Run the server manually to see errors:

   ```
   cd ~/projects/google-workspace-mcp && node scripts/start.js
   ```

4. Check OAuth credentials exist

OAuth "This App Isn't Verified" Warning

This is expected. Since your Google Cloud project is in "Testing" mode and not verified by Google, you will see this warning. Click **Advanced** → **Go to MCP Workspace Server (unsafe)** to proceed. This only appears because you haven't submitted the app for Google verification, which is unnecessary for personal use.

OAuth "Access Blocked: org_internal"

Fix: Change your OAuth consent screen from "Internal" to "External". The "Internal" option only works for Google Workspace organisation accounts, not personal @gmail.com accounts.

Token Expiry / Re-auth Required

Google OAuth tokens expire periodically. If you get authentication errors:

```
cd ~/projects/google-workspace-mcp
node scripts/auth-utils.js clear
```

Then restart OpenCode. The next tool call will trigger a fresh OAuth flow.

MCP Tools Not Appearing in OpenCode

Fix:

1. Run `opencode mcp list` to see connected servers
2. Check the MCP server is enabled in your config (`"enabled": true`)
3. Restart OpenCode after config changes
4. Check for error messages in the OpenCode logs

AI Gateway Shows No Logs

Fix:

1. Verify you are routing through the gateway (check the `baseURL` in your provider config)
2. The gateway ID in the URL must match exactly
3. Check that logging is enabled in the gateway settings

Neuron Limit Reached

Symptom: "429 Too Many Requests" or "Neuron limit exceeded"

Fix:

1. Wait until midnight UTC for the daily reset
2. Switch to a smaller model (8B instead of 70B)
3. Enable caching in AI Gateway
4. Consider upgrading to Workers Paid ($5/month) for unlimited neurons

---

16. Reference Links

Cloudflare Documentation

- Build a Remote MCP Server
- MCP Transport
- MCP Authorization
- MCP Tools
- AI Gateway MCP Server
- Workers AI Models
- Workers AI Pricing
- AI Gateway
- Workers
- Secure MCP Servers with Access

OpenCode

- OpenCode Website

- OpenCode GitHub

Google Workspace MCP Server

- GitHub Repository

Cloudflare Blog

- Enterprise MCP
- Model Context Protocol