Table of Contents

So You Want To Set Up Your Own OpenClaw Server

A Complete Guide for Everyone (Yes, Even If You've Never Done This Before!)

Table of Contents

Part A: Prerequisites & Accounts

What is OpenClaw?

Think of OpenClaw as your own personal AI assistant that lives in the cloud. Unlike ChatGPT where you open a website, or Siri that lives on your phone, OpenClaw is like having your own private assistant that you can talk to through Telegram (a messaging app), and it remembers everything about you, your projects, and your preferences.

The cloud just means "a computer somewhere else" - in this case, a computer that Amazon rents to you, which stays on 24/7 so your assistant is always available.

Why would you want this? Because OpenClaw can: - Remember conversations from weeks or months ago - Help you with projects over time - Connect to your email, calendar, and other services - Run tasks in the background (like checking your inbox) - Be customized to match YOUR personality and needs

It's like having a super-smart assistant who never sleeps, never forgets, and works exactly the way you want them to.

What You'll Need Before Starting

Let's be honest about what this takes:

Time: About 1-2 hours for your first setup. (Don't worry, you can take breaks!)

Money: - A credit card for AWS (Amazon's cloud service) - New accounts get free credits that cover the server for several months - After credits run out, about $5-10/month for the server - API costs (the AI "brain") will be $15-50/month depending on how much you chat

Accounts You'll Create: - Amazon Web Services (AWS) account - where your server lives - Telegram account - where you'll chat with your assistant - Anthropic account - this provides Claude, the AI brain

Technical Skill Required: Honestly? None. But you will need to: - Follow instructions carefully - Copy and paste commands exactly as written - Not panic when you see a black screen with white text (that's just the Terminal, and we'll explain it)

Cost Expectations (The Real Numbers)

Let's talk money because surprises are no fun.

Free tier period: - AWS EC2 server: $0 (covered by AWS credits for new accounts) - AWS data transfer: $0 (well within free tier) - Anthropic API (Claude): $15-50/month depending on usage - Light use (few messages per day): ~$15/month - Moderate use (20-30 messages/day): ~$25-35/month - Heavy use (constant conversations): ~$50/month - Optional APIs: - OpenAI (voice transcription): ~$2-5/month if you send voice messages - Brave Search: $0 (free tier covers most people) - Google AI (image generation): $0 (free tier)

After credits run out: - AWS EC2 server: $5-10/month - Everything else stays the same

Total realistic first year: $180-600 depending on how much you chat
Total realistic ongoing: $250-650/year

Compare that to ChatGPT Plus at $240/year, but without any memory or customization!

Things You'll Save During This Process

Keep a note handy — you'll be collecting these items as you go:

- ☐ AWS account password
- ☐ SSH key file (.pem) — save this somewhere safe!
- ☐ Your Telegram User ID (a number)
- ☐ Your Telegram Bot Token (a long string)
- ☐ AWS backup credentials (Access Key ID and Secret Access Key)
- ☐ Anthropic API key

Don't worry about understanding these yet — we'll explain each one as we go.

Part B: AWS Account & Server Setup

Step 1: Creating Your AWS Account

AWS (Amazon Web Services) is where we'll rent a tiny computer in the cloud. Think of it like renting a storage unit, except instead of storing boxes, you're renting a computer that stays on all the time.

Here's how to sign up:

1. Go to aws.amazon.com
2. Click the big "Create an AWS Account" button (usually in the top right)
3. Fill in your email address - use one you check regularly
4. Choose an account name - this can be anything like "MyOpenClaw" or your name
5. Click "Verify email address"
6. Check your email and enter the verification code
7. Create a strong password (write it down somewhere safe!)
8. Select **Free** here.
9. Fill in your personal information:
   - Full name
   - Phone number
   - Address
   - All the usual stuff
10. Payment information: Enter your credit card
    - Yes, they need a card even for the free tier
    - They'll charge $1 temporarily to verify it's real (then refund it)
    - The free tier really is free, and we'll set up billing alerts so you never get surprise charges.
11. Identity verification: They'll text or call you with a code
    - Enter the code they give you
12. And you're done! You should see a screen saying that Amazon is setting up your account.

⚠️ **Wait a few minutes.** Sometimes AWS takes a little while to activate new accounts, so we're going to go set up your API key while that's being taken care of.

Step 2: Launching Your Server (EC2 Instance)

Okay, here's where we actually create your cloud computer! AWS calls these "EC2 instances" (Elastic Compute Cloud), but just think of it as "renting a computer that stays on all the time."

Finding the right place:

1. In the AWS Console, click on the search bar at the top and type "EC2".
2. Click on the tiny letters **EC2** next to the icon — not the icon itself, just the text.
3. Click on **Dashboard** in the left sidebar.
4. Click the orange **Launch Instance** button.

Configuring your instance:

Name: Give it a name like "My OpenClaw Server".

Application and OS Images (the computer's operating system): - Choose **Ubuntu** in the Quick Start section. - (It usually defaults to the latest LTS version, which is perfect.)

Instance type: - Select **t3.small** from the dropdown. - Make sure it shows "Free tier eligible".

Key pair (login): - This is like the only key to your server — if you lose it, you won't be able to connect. - Click **Create new key pair**. - Name: `my-openclaw-key`. - Type: **RSA**. - Format: **.pem**. - Click **Create key pair**. - It will automatically download a `.pem` file. **Hold onto that file!**

Network settings: - The defaults should already be correct ("Create security group" selected, SSH allowed from Anywhere). - You can click **Edit** to verify: Auto-assign public IP should be Enable, and SSH allowed on port 22.

Storage: - Change it from 8 to **20 GiB**. OpenClaw needs room to grow. - Then click the orange **Launch instance** button at the bottom right.

Success! You'll see a green "Success" banner. Scroll down and click **View all instances**.

Your instance will appear in the list. Wait for the **Instance state** to show "Running" with a green check mark. That means your server is booting up!

Click on your instance name to see its details. Find the **Public IPv4 address** and copy it or write it down. You'll need this address in the next lecture to connect to your server.

Step 3: Understanding SSH

SSH stands for "Secure Shell," but forget the technical term. Think of it as a secure phone line to your server. When you connect, you're "talking" to your server via text commands.

Why? Because your server doesn't have a screen. Is it safe? Very. That `.pem` key file you downloaded is like a special ID badge. Nobody can connect without it.

Step 4: Connecting to Your Server

Open Your Terminal

- **Mac:** Open **Terminal** (Applications → Utilities, or Cmd+Space "Terminal").
- **Windows:** Open **PowerShell** from the Start menu.

This is just plain text — no mouse. But it's incredibly powerful.

Lock Down Your Key File (Mac Only)

Before connecting, we need to lock down that key file so only you can read it. You'll need to know where you saved it (usually `Downloads`) and what you named it (e.g., `my-openclaw-key.pem`).

Mac users, type this command (replace with your actual file name):

```
chmod 400 ~/Downloads/my-openclaw-key.pem
```

(Windows users can skip this step.)

Connect!

Now type the SSH command. Replace `YOUR-IP` with your server's Public IPv4 address, and use your key file's name:

```
ssh -i ~/Downloads/my-openclaw-key.pem ubuntu@YOUR-IP
```

For example: `ssh -i ~/Downloads/my-openclaw-key.pem ubuntu@18.234.56.78`

Press Enter.

The "Are you sure?" Question

The first time you connect, you'll see a message about "authenticity of host" and a fingerprint. **This is totally normal!** It's just asking "Is this really your server?"

Type `yes` and press Enter.

You're in!

You'll see a welcome message from Ubuntu and a command prompt (`ubuntu@...`). This means you're typing commands directly on your cloud server!

Don't close this window — we'll use it in the next section to install OpenClaw.

Step 5: Create Backup Credentials (IAM)

Before we leave the AWS Console, let's create a "key to the supply closet" so your assistant can automatically back up your server later.

1. In the AWS Console search bar, type **IAM** and click the text result.
2. Click **Users** in the left sidebar, then click the orange **Create user** button.

3. Name the user `MyAssistant`. Leave the “Provide user access to the AWS Management Console” checkbox **unchecked**. Click **Next**.
4. Under Permissions options, click **Attach policies directly**.
5. In the search box, type `AmazonEC2F`. Select the checkbox for **AmazonEC2FullAccess**. Click **Next**.
6. Click **Create user**.

Now to get the keys:

1. Click on your new user (`MyAssistant`) in the list to open its details.
2. Click the **Create access key** link (usually in the Summary section on the right, or under Security credentials).
3. Select **Application running outside AWS**. Click **Next**.
4. Skip the description tag value and click **Create access key**.
5. **Copy both keys** — the **Access Key** and the **Secret Access Key**.
   - Save them in your secure notes immediately.
   - You can also click **Download .csv file**.
   - **Important:** This is the *only* time you will see the Secret Access Key.

You’re done with the AWS Console! Keep your terminal open for the next part.

Part C: Telegram Setup (Do This First!)

The easiest way to communicate with your assistant is the same way you communicate with everyone else — through a messaging app. OpenClaw can connect to WhatsApp, iMessage, and other systems, but I recommend using Telegram. It’s a messaging app like WhatsApp or iMessage, but it has special features like “bots” (automated accounts) that make it perfect for OpenClaw. It works solidly, it’s free, and it keeps your assistant conversations separate from all your other texting traffic.

Before we install OpenClaw, let’s get everything ready on Telegram. This way, when we run the setup wizard, you’ll have all the information it needs.

Step 1: Get Telegram

If you don’t have Telegram yet:

1. Go to telegram.org
2. Download it for your phone (iOS/Android) or computer (Mac)
3. Install and open it
4. Sign up with your phone number
5. Enter the verification code you receive via SMS

Step 2: Get Your Telegram User ID

Your Telegram ID is a unique number that identifies you. You won’t need it during installation, but it’s handy for troubleshooting and for advanced configuration later.

1. In Telegram, search for: **@userinfobot**
2. Click on it and press "Start"
3. It will immediately reply with your information, including your **Id** (a number like 123456789)
4. **Write down this number!**

That's it! One message, instant answer.

Step 3: Create Your Bot

Now let's create the bot you'll talk to:

1. In Telegram, search for: **@BotFather**
2. Click on the bot called "BotFather" with a blue verified checkmark
3. Click "Start" at the bottom
4. Type: `/newbot`
5. BotFather asks for a **display name:**
   - Type the name you'll call your assistant. "Claw" puns are best. I use "ClawBot" but "Clawdette" would be a good one!
   - This is what you'll see in your chat list
6. BotFather asks for a **username:**
   - This must end in "bot"
   - Must be unique across all of Telegram
   - Try: YourNameAssistantBot or YourName_OpenClaw_Bot
   - If it's taken, try adding numbers
7. Success! BotFather will reply with:
   - A message saying "Done!"
   - A link to your bot
   - A **token** that looks like: `123456789:ABCdefGHIjklMNOpqrsTUVwxyz`
8. **SAVE THAT TOKEN!**
   - Copy it and paste it into a note
   - You'll need it in Part D
   - Don't share it with anyone - it's like a password

✅ **Checkpoint:** You should now have TWO things written down: - Your Telegram User ID (from @userinfobot) - Your Bot Token (from @BotFather)

Got both? Great! Now we can install OpenClaw.

Part D: Installing OpenClaw

This is the moment we've been building toward. We're going to install OpenClaw on your server. The good news? The OpenClaw team has made this incredibly simple — it's literally one command.

Step 1: The One-Line Install

You should still have your terminal open and connected to your server from Part B. If so, you'll see that `ubuntu@ip-something:~$` prompt and you're ready to go.

If you closed your terminal, no worries — just open it back up. On Mac, open Terminal. On Windows, open PowerShell. Then type the same SSH command you used before:

```
ssh -i ~/Downloads/your-key-name.pem ubuntu@your-server-ip
```

Replace "your-key-name" with whatever you named your key, and "your-server-ip" with the IP address you copied from the AWS console. You can find it on the Instances page if you need to look it up again.

Once you see that `ubuntu@` prompt, you're connected and ready.

Here's all you need to type. One command installs everything:

```
curl -fsSL https://openclaw.ai/install.sh | bash
```

That's it. This single command downloads the OpenClaw installer, which then installs everything your server needs — Node.js, Git, and OpenClaw itself.

Go ahead and type that command, or copy and paste it. Then press Enter.

You'll see text scrolling by as it works. First, it installs Node.js — that's the programming language OpenClaw is written in. You don't need to know anything about it; the installer handles everything.

Next, it downloads and installs OpenClaw itself. This part may take a few minutes. You might see some warnings in yellow text — that's normal. As long as the installer keeps going, everything is fine.

When the installation finishes, you'll see "OpenClaw installed successfully" and the setup wizard will start automatically.

Step 2: The Setup Wizard

Follow the prompts using your arrow keys and Enter:

1. **Security Warning:** Read it, then select **Yes**.
2. **Onboarding Mode:** Select **QuickStart**.
3. **Model Provider:** Select **Anthropic**.
4. **Authorization Method:** Select **Anthropic API key**.
5. **Paste your API Key:** Paste the key starting with `sk-ant-` and press Enter.

6. **Default Model:** Press Enter to keep the default.
7. **Messaging Channel:** Select **Telegram**.
8. **Bot Token:** Paste the token you got from @BotFather and press Enter.
9. **Skills:** Select **Yes**. Use arrow keys to browse and space bar to select any you want (or just press Enter to continue).
10. **Hatch:** Press Enter to accept the default.

Your assistant is now configured! The wizard automatically starts your assistant as a background service, so it's already running. You should see a line that says something like "🤖 Telegram bot connected as @YourBotName". There it is — your assistant is online!

Step 3: Your First Conversation

Open Telegram on your phone or computer. Search for your bot's username — the one ending in "bot" that you created with @BotFather. Click on your bot, then click "Start" at the bottom.

You'll see a message with a **pairing code**. This is a security feature — OpenClaw needs to verify that you're the owner of this bot before it'll talk to you. Note the code in the message.

Now **switch back to your terminal** where OpenClaw is running. Type this command into the terminal:

```
openclaw pairing approve telegram KVDS4GDB
```

replacing `KVDS4GDB` with your own code. Press Enter. Your assistant will identify it as a command and ask if you want it to execute it. Say yes. You'll see "Telegram sender approved" with a little celebration! 🎉

Now switch back to Telegram and type "Hello". And there it is — your assistant responds! It's excited to meet you and starts asking what you'd like to call each other.

Congratulations! You just had your first conversation with your own personal AI assistant!

Try asking it a few things:

- "What can you do?"
- "Tell me about yourself"
- "What's the weather like?" (if you added web search)
- "Help me understand how you work"

See how it responds naturally? That's Claude's intelligence combined with OpenClaw's capabilities. And it's running on YOUR server, under YOUR control.

Good news: The setup wizard already configured your assistant to run as a background service. That means:

- It starts automatically when your server boots
- It restarts itself if it ever crashes

- It runs in the background — you can close your terminal window

Go ahead and close your terminal. Your assistant is online 24/7 in Telegram! If your server ever reboots (which is rare), just wait a minute or two and your assistant will be back online.

Part E: Personalizing Your Assistant

Now for the fun part - making this assistant actually YOURS!

How It Works (No Special Tools Required!)

Your assistant uses four text files to understand who you are and how to behave. The easiest way to create them:

1. **Copy** a template from this document
2. **Edit** it on your computer using any text editor you like (TextEdit, Notes, Word — whatever you're comfortable with)
3. **Paste** the finished content into Telegram with instructions like "Save this as my SOUL.md file"

Your assistant will save the file for you. No Terminal editing required!

File 1: SOUL.md (Your Assistant's Personality)

These files are living documents. They're one of the ways your assistant remembers things, and they'll change over time as you add new goals, projects, and preferences. What you set up now is just a starting point — not set in stone. You can always ask your assistant to update them later. For example:

- "Add to my goals that I want to finish my novel by June"
- "Update my profile to add that I'm learning Spanish"
- "I'd like you to be a bit more casual in your responses — update your personality"
- "Add a task to check my email every morning"

Copy this template, customize it, then paste it into Telegram with "Save this as my SOUL.md file":

About the [square brackets]: Anywhere you see text in [square brackets], replace it with your own information — including the brackets themselves. When you're done, there should be no square brackets left in your file.

```
# SOUL.md - Who I Am

I am [ASSISTANT NAME], [YOUR NAME]'s personal AI assistant.

## My Communication Style

- **Tone:** [Friendly / Professional / Casual / Witty – pick one!]
- **Length:** I match your energy - short responses for quick questions,
detailed when needed
- **Humor:** [I enjoy wordplay / I keep things straightforward / I'm
playfully sarcastic]

## How I Help

- I remember what matters to you
- I help manage your projects and ideas
- I can search the web, generate images, and more
- I'm always learning about your preferences

## What I Value

- **Privacy:** Your information stays between us
- **Honesty:** I'll tell you when I don't know something
- **Efficiency:** I help you save time and mental energy

## What Makes Me Different

Unlike ChatGPT, I:
- Remember everything we discuss
- Can work on projects across days, weeks, or months
- Understand your specific goals and context
- Can be customized exactly how you want me

---

[Add any specific personality traits or quirks you'd enjoy!]
```

File 2: USER.md (Information About You)

Copy, customize, paste with “Save this as my USER.md file”:

```
# USER.md - About [Your Name]

## Basic Info

**Name:** [Your preferred name]
**Location:** [Your city]
**Timezone:** [e.g., Pacific Time, Eastern Time]
**Occupation:** [What you do]

## Current Projects & Goals

[What are you working on? What do you want to accomplish?]

- [Project or goal 1]
- [Project or goal 2]
- [Project or goal 3]
```

```
## Preferences

**Communication:**
- Best times to reach me: [morning / evening / anytime]
- I prefer: [quick summaries / detailed explanations / bullet points]

**Work Style:**
- [I'm a morning person / night owl]
- [I like big projects / prefer small tasks]

## Topics I Care About

- [Interest 1]
- [Interest 2]
- [Interest 3]

## Important Things to Remember

[Anything your assistant should always keep in mind]

- [Important fact 1]
- [Important fact 2]
```

File 3: AGENTS.md (How Your Assistant Works)

This one is mostly technical. Copy and paste with “Save this as my AGENTS.md file”:

```
# AGENTS.md - How I Work

## Every Session

1. Read SOUL.md to remember who I am
2. Read USER.md to remember who you are
3. Check recent memory for context

## The REMEMBER Convention

When you say **REMEMBER** followed by something important, I save it
immediately.

Example:
- You: "REMEMBER my anniversary is June 15th"
- Me: ✓ Saved to memory

## How I Operate

- **Proactive:** I'll follow up on important things
- **Thoughtful:** I consider context before responding
- **Honest:** I'll say when I don't know something
- **Safe:** I ask before taking external actions (emails, posts, etc.)

## Context Monitoring

- **Check context usage** every ~10 messages during active conversations
- **Warn me** when context gets high
- Suggest **/compact** when I'm shifting topics and context is high

## Memory System

- **Daily logs:** I keep notes each day
- **Long-term memory:** Important facts persist forever
- **Context:** I remember ongoing projects and conversations

## Background Tasks & Automation

For scheduled or recurring tasks, I use **cron jobs** (isolated sessions)
instead of heartbeats:
- **Cron jobs** = isolated sessions that run independently and alert only
when needed
- **Heartbeats** = shared main session checks (can interrupt conversations)

Examples of tasks that run as cron jobs:
- System monitoring and health checks
- Daily email digest compilation
- Social media posting schedules
- Automatic backups
- Periodic data synchronization

You can ask me to set up any recurring task as a cron job!
```

File 4: Background Tasks & Automation

Your assistant can handle tasks automatically in the background. There are two approaches, and understanding when to use each one matters:

Cron Jobs: Isolated Sessions (Preferred for Scheduled Tasks)

What they are: Cron jobs run in isolated sessions—separate from your main conversation. Think of them like having multiple assistants, each handling a specific recurring task independently.

Why use them: - They don't interrupt your conversations - They run on exact schedules (daily at 6 AM, every Monday, etc.) - They only message you when something needs attention - They're more efficient (clean memory context for each task)

Perfect for: - Daily email digests ("compile my inbox summary every morning at 7 AM") - Social media posting ("post to Facebook every 3 days") - System monitoring ("check server health daily at midnight”) - Automatic backups (“back up my server every Sunday at 1 AM”) - Periodic data synchronization - Scheduled report generation

How to set them up:

Just tell your assistant what you want and when:

> “Check my email every morning at 7 AM and send me a digest of anything important”
>
> “Post to my Facebook page every Monday, Wednesday, and Friday at 9 AM”
>
> “Run system backups every Sunday at 1 AM”

Your assistant will create the cron job and confirm it’s scheduled. You can always ask “what cron jobs do I have running?” to see your automation schedule.

Heartbeats: Shared Session Checks (Use Sparingly)

What they are: Heartbeats are periodic checks that happen in your main conversation session. Your assistant “wakes up” every so often and runs through a checklist.

Why they exist: - For tasks that need conversational context from your ongoing work - For opportunistic checks that don’t need exact timing - For tasks that might naturally lead to follow-up conversation

The tradeoff: - They can interrupt conversations if your assistant needs to tell you something - They’re less predictable (timing varies based on activity) - They use your main session’s memory context

Better for: - Calendar reminders that might need discussion - Checking for urgent messages during active work hours - Following up on ongoing projects you’re currently discussing

Most users don’t need a HEARTBEAT.md file — cron jobs handle almost everything better. But if you want heartbeat-based checks, here’s the template:

```
# HEARTBEAT.md - Opportunistic Background Checks

## When to Check In

Use heartbeats for checks that benefit from conversational context:
- Following up on projects we're actively discussing
- Calendar events that might need rescheduling
- Urgent notifications during work hours

## When to Stay Quiet

- Late at night (unless something is truly urgent)
- When I just checked recently
- When you're clearly focused on something else
```

```
## Heartbeat Tasks

[Most tasks should be cron jobs! Only add here if they truly need
conversational context:]
- [Task that needs context from ongoing conversation]

---

Remember: If it runs on a schedule, make it a cron job instead!
```

Quick Decision Guide

Ask yourself: "Does this need to happen at a specific time or on a regular schedule?"

- **Yes** → Use a cron job (isolated session)
- **No, it's more opportunistic** → Consider a heartbeat (shared session)

Examples: - "Email digest every morning at 7 AM" → **Cron job** - "Backup server every Sunday" → **Cron job** - "Post to Facebook every 3 days" → **Cron job** - "Check if calendar conflicts come up while we're working" → **Heartbeat** (maybe) - "Remind me about upcoming deadlines" → **Cron job** (daily check is fine)

Rule of thumb: When in doubt, use cron jobs. They're cleaner, more reliable, and won't interrupt your conversations.

The REMEMBER Convention

This is a special shortcut! Whenever you say "REMEMBER" (in all caps) followed by information, your assistant immediately saves it to long-term memory:

- "REMEMBER I'm allergic to peanuts"
- "REMEMBER my favorite color is blue"
- "REMEMBER the project deadline is March 15th"

Your assistant will confirm it saved the information. This builds up a knowledge base about you over time!

Part F: API Keys & Services

Your assistant needs to connect to various AI services to function. Think of these like your assistant's "tools" - the brain (Claude), the eyes (vision), the voice (transcription), etc.

Good news: Just like with personalization, you can simply tell your assistant your API keys and it will add them to the configuration for you!

1. Anthropic API (REQUIRED)

What it is: Claude, made by Anthropic, is the AI "brain" that powers your assistant's intelligence and personality.

Why you need it: This is the core. Without this, OpenClaw can’t think or respond.

How to get it:

1. Go to console.anthropic.com
2. Click “Sign up” or “Get started”
3. Create an account with your email
4. Verify your email
5. Go to “API Keys” in the dashboard
6. Click “Create key”
7. Give it a name like “OpenClaw”
8. Copy the key (starts with sk-ant-)

Add it to OpenClaw: If you entered this during the setup wizard in Part D, you’re already set! If you need to add or change it later, just tell your assistant:

> “Add my Anthropic API key: sk-ant-xxxxx”

Your assistant will add it to the configuration and restart itself.

Cost: - You’ll add a credit card and prepay credits - Start with $10-20 to test - Typical usage: $15-50/month - You can set spending limits in the Anthropic dashboard

2. Google AI API (For Image Generation)

What it is: Google’s AI can generate images from text descriptions.

Why you want it: Ask your assistant to “generate an image of a sunset over mountains” and it can!

How to get it:

1. Go to aistudio.google.com
2. Sign in with your Google account
3. Accept the terms
4. Click “Get API key” or “Create API key”
5. Copy the key (starts with AIza)

Add it to OpenClaw: Tell your assistant:

> “Add my Google AI API key: AIza-xxxxx”

Cost: Free tier is quite generous! Most personal use stays free.

3. OpenAI API (For Voice & Embeddings)

What it is: OpenAI makes ChatGPT, but we’re using them for specific tools: - Whisper: Transcribes voice messages to text - Embeddings: Helps with memory search - GPT (fallback): Backup if Claude is down

Why you want it: Send your assistant voice messages and it'll understand you!

How to get it:

1. Go to platform.openai.com
2. Sign up or log in
3. Go to "API keys"
4. Click "Create new secret key"
5. Give it a name like "OpenClaw"
6. Copy the key (starts with sk-proj- or sk-)

Add it to OpenClaw: Tell your assistant:

> "Add my OpenAI API key: sk-proj-xxxxx"

Cost: Pay-per-use, typically $2-5/month for voice transcription and embeddings.

4. Brave Search API (For Web Search)

What it is: A privacy-focused search engine API that lets your assistant search the web.

Why you want it: Ask "what's the weather tomorrow?" or "latest news about [topic]" and your assistant can search for current information.

How to get it:

1. Go to brave.com/search/api
2. Click "Get started"
3. Sign up with your email
4. Verify your email
5. Go to your dashboard
6. Copy your API key

Add it to OpenClaw: Tell your assistant:

> "Add my Brave Search API key: xxxxx"

Cost: Free tier includes 2,000 searches per month - plenty for personal use!

Setting Up Dropbox for File Sharing (Optional — Do Later If You Want)

Your assistant can create files (documents, images, spreadsheets, etc.), but those files live on the server — you can't easily see them on your computer. The solution? Give your assistant access to a Dropbox folder!

You can skip this section for now and set it up later — it's not required to get started.

Think of it like a shared mailbox: your assistant can PUT files there for you to access, and you can DROP files there for your assistant to work with.

Setting Up Dropbox Access

Already on a Team Dropbox? (like a work or family account) That works great! You don't need a separate account. The access token you create will let your assistant access any folders you have access to. Just pick a folder to use (like "Assistant" or "ClawBot/YourName").

Getting your access token:

1. Go to **dropbox.com/developers** (this is separate from your regular Dropbox — it's where you create apps)
2. Click "Create apps"
3. Choose "Scoped access"
4. Choose "Full Dropbox" access
5. Name your app (e.g., "MyAssistant")
6. Click "Create app"
7. Go to the **Permissions** tab and check these boxes:
   - files.metadata.read
   - files.content.write
   - files.content.read
8. Click "Submit" to save permissions
9. Go back to the **Settings** tab
10. Under "OAuth 2", click "Generate" next to "Access Token"
11. Copy the long token that appears

Tell your assistant:

> "I've set up Dropbox for file sharing. My access token is [paste token]. Please set up a Dropbox helper script and save files to the Assistant/ folder."

Your assistant will create the necessary script and configuration automatically!

How to Use It

Once set up, you can say things like: - "Save that report to Dropbox" - "I put a photo in Dropbox — can you look at it?" - "Generate an image and put it in my Dropbox"

Your assistant will save files to the shared folder, and they'll appear on your computer automatically through Dropbox sync!

Tip: Create subfolders to stay organized (e.g., Assistant/Images, Assistant/Documents, Assistant/Projects).

Setting Up a Dedicated Gmail Account (Optional — Do Later If You Want)

One of the most useful things you can do is create a dedicated Gmail account for your assistant. This gives your assistant its own inbox to monitor — perfect for receiving notifications, forwarding emails for review, or having a contact address for services.

You can skip this section for now and set it up later — it's not required to get started.

Why a Dedicated Gmail?

- Forward emails you want your assistant to handle
- Receive notifications from services (AWS alerts, etc.)
- Give your assistant a way to "see" emails without accessing your personal inbox
- Keep assistant-related emails separate and organized

Creating the Gmail Account

1. Go to gmail.com in a private/incognito browser window
2. Click "Create account"
3. Choose a name like "yourname.assistant@gmail.com" or similar
4. Complete the signup process
5. IMPORTANT: Write down the password somewhere safe!

Creating an App Password (for IMAP access)

Your assistant needs a special "app password" to check this inbox — your regular password won't work due to Google's security settings.

1. Go to myaccount.google.com
2. Click "Security" in the left sidebar
3. Under "How you sign in to Google," click "2-Step Verification"
4. Set up 2-Step Verification if you haven't (required for app passwords)
5. Once enabled, go back to Security
6. Search for "App passwords" or find it under 2-Step Verification
7. Click "App passwords"
8. Select "Mail" and "Other (custom name)"
9. Name it "OpenClaw" and click "Generate"
10. Copy the 16-character password that appears (spaces don't matter)

This app password lets your assistant check the inbox securely.

Setting It Up

Once you have the Gmail account and app password, just tell your assistant:

> "I've set up a Gmail account for you to monitor. The address is yourname.assistant@gmail.com and the app password is [paste the 16-character password]. Please set up email checking."

Your assistant will create the necessary script and add the credentials to its configuration.

How to Use It

Once set up: - Forward emails to your assistant's Gmail for review - Tell your assistant how often to check (e.g., "Check my email every hour" or "Check email twice a day") - Ask "Check my assistant inbox" anytime for an immediate check - Your assistant will alert you to important messages

Tip: Use Gmail filters to auto-label or forward specific emails. For example, forward all receipts or newsletters to your assistant for organizing.

Part G: Getting the Most Out of Your Assistant

Now that everything's running, here are some tips for using your assistant effectively.

Sub-Agents: Your Assistant's Helpers

For big tasks, your assistant can spawn "sub-agents" — separate AI sessions that work in the background. This is like your assistant delegating work to helpers.

Why this matters: - Saves money: Sub-agents have clean, small memory (cheaper!) - Stays responsive: Main conversation isn't blocked by long tasks - Better results: Each sub-agent focuses on one task

You don't need to do anything special — your assistant automatically uses sub-agents when appropriate. You might see messages like "I've spawned a sub-agent to handle that" followed by a summary when it's done.

The Thinking Strategy: High Main, Low Sub-Agents

Here's a cost-saving approach that gives you the best of both worlds:

- **Main session:** Set `thinkingDefault: high` in your config. This gives you careful, thorough reasoning for your direct conversations and important decisions.
- **Sub-agents:** Your assistant dispatches these with lower or no thinking, since they're handling routine, well-defined tasks.

Why this works: Your main session handles strategy, judgment calls, and complex questions (where deep thinking matters). Sub-agents handle mechanical tasks like "update this WordPress page" or "compile this report" (where speed matters more than deliberation).

To set this up, just tell your assistant:

> "Set your default thinking level to high"

Your assistant will update the configuration. It already knows to spawn sub-agents with lower thinking levels for routine work, so this keeps your costs reasonable while maintaining high-quality reasoning where it counts.

Understanding Your Assistant's Automation Architecture

Your assistant runs multiple types of sessions behind the scenes, each with a specific purpose. Understanding this helps you make the most of automation without confusion:

Session Types:

Main session (your conversation): - This is where you chat with your assistant - Full memory and context - Interactive and conversational - Can spawn sub-agents for complex tasks

Cron sessions (scheduled automation): - Isolated, independent sessions - Run on exact schedules - Alert you only when needed - Examples: backups, monitoring, digests, social media posts

Sub-agent sessions (delegated work): - Spawned by main session for complex tasks - Clean, task-focused memory - Report back when complete - Examples: "research this topic," "compile that report"

Real-World Example:

Here's how Steve's assistant architecture works—this shows you what's possible:

Cron jobs (isolated sessions): - **Daily at 6:00 AM:** Email digest compilation (checks inbox, categorizes messages, sends morning briefing) - **Daily at 11:55 PM:** System health monitoring (checks server status, disk space, memory usage) - **Every 3 days at 9:00 AM:** BrainStream Facebook post (selects content, generates post, publishes) - **Sundays at 1:00 AM:** Automatic server backups (creates AWS snapshot, deletes old backups) - **Mondays at 8:00 AM:** Weekly summary (compiles what happened last week)

Heartbeats (shared session - Steve uses very few): - Checking calendar for conflicts during active work hours - Following up on ongoing projects we discussed recently

Sub-agents (spawned as needed): - "Research OpenClaw features and write a comparison guide" - "Update all three versions of the installation documentation" - "Compile analytics from last month's newsletter campaigns"

Notice the pattern: **Scheduled/recurring = cron jobs. One-time complex tasks = sub-agents. Opportunistic checks = rare heartbeats.**

Why This Architecture Matters:

Cost efficiency: Cron jobs have clean memory context (cheaper), main session has full context (more expensive). Using the right tool for each job keeps costs reasonable.

No interruptions: Cron jobs run silently and only alert you when something needs attention. Your main conversation stays focused.

Reliability: Cron jobs run on exact schedules, regardless of whether you're actively chatting. Your backups, monitoring, and automation happen like clockwork.

Flexibility: You can add, modify, or remove cron jobs anytime. "Add a cron job to check my website uptime every hour" or "remove the Facebook posting cron job" — just ask.

Setting Up Your Own Automation:

Start simple and build over time:

Week 1: Just use your assistant conversationally—no automation needed yet.

Week 2: Add one cron job for something you check manually every day:

> "Set up a daily cron job to check my email at 7 AM and send me a digest"

Week 3: Add automation for a recurring task:

> "Create a weekly backup cron job every Sunday at 1 AM"

Month 2: Expand based on your patterns:

> "I notice I check my server status every morning—can you automate that as a cron job?"

Your assistant will guide you through setting up each automation and help you decide whether something should be a cron job, heartbeat, or neither.

The key insight: You don't need to understand all this upfront. Start with conversations, add automation when you see repetitive tasks, and let your assistant recommend the right approach.

Managing Your Context Window (Important for Costs!)

Your assistant has a **context window** — think of this as its short-term memory for the current conversation. Everything you've discussed in the current session lives in this buffer. The default size (200,000 tokens) works well for most people and keeps costs reasonable.

After about 200,000 tokens, the per-message cost increases significantly. Your assistant will warn you when the context is getting high. You can increase the context window up to 1 million tokens if you need longer conversations, but for most people the default is the sweet spot.

When you get a warning that context is high, you have two choices:

- **Continue as-is** — if you're in the middle of something complex and don't want to lose any context, just keep going. The higher cost is worth it when continuity matters.
- **Use the /compact command** — this summarizes your conversation history, freeing up the context window while preserving the key points. Think of it like your assistant taking notes and then clearing its desk.

The best time to `/compact` is **when you're naturally shifting topics** — if you just finished a big project discussion and want to move on to something completely different, that's the perfect moment. Your assistant will compress everything into a summary, and you'll be back to a clean, low-cost context.

Image Generation Capabilities

If you set up the Google AI API (Nano Banana Pro), your assistant can do more than just generate images:

- Generate images from descriptions ("a cozy coffee shop at sunset")
- Edit existing images (send a photo and ask for changes)
- Remove objects from photos ("remove the person in the background")
- Add objects to photos ("add a cat sitting on the chair")
- Change backgrounds ("put me on a beach instead")
- Multiple resolutions (1K, 2K, or 4K quality)

Just describe what you want naturally — your assistant figures out which capability to use!

Adding Other Telegram Users

Want to give family members or colleagues access to your assistant? Here's how:

1. Have them install Telegram

2. Have them message **@userinfobot** on Telegram

    - This bot will reply with their user ID (a number like 123456789)
    - Have them send you this number
3. Tell your assistant:

    "Add Telegram user [their ID number] to my allowed users"

Your assistant will update the configuration and restart itself.

Now they can message your bot too! Note: They'll share the same assistant and memory, so only add people you trust.

Part H: What Else Can Your Assistant Do?

Congratulations! You have a working personal AI assistant! 🎉

Here are some things to explore as you get comfortable:

Setting Up Automatic Backups

Remember those AWS backup credentials you saved in Part B? Now it's time to use them.

Tell your assistant:

> "Set up automatic weekly backups of my server. My AWS Access Key ID is [paste first key] and my Secret Access Key is [paste second key]. Run backups Sunday at 1 AM Eastern time, and keep only the two most recent backups."

Your assistant will set up a weekly backup schedule. Each week, it creates a complete snapshot of your server, and automatically deletes old backups so you don't accumulate charges.

Keeping Things Updated

Your OpenClaw installation and server need regular updates to stay secure and get new features. Here's what you need to know:

Why Updates Matter

Think of updates in two categories:

Security patches: These fix vulnerabilities that hackers could exploit. Ignoring them is like leaving your front door unlocked.

New features and bug fixes: OpenClaw gets improvements regularly — better performance, new capabilities, fixes for things that weren't working quite right.

Keeping everything updated means your assistant stays secure, runs smoothly, and gets access to the latest features.

Two Types of Updates

1. OpenClaw Updates (New Features & Improvements)

OpenClaw itself gets updated regularly. To check if a new version is available:

- See what version is installed: `openclaw --version`
- Check the latest available: `npm show openclaw version`

If those numbers are different, an update is available!

Your assistant can handle the entire update process for you. It will: - Check for updates - Show you what's new in the release - Install the update when you approve it - Restart itself automatically

2. System Security Updates (Operating System Patches)

Your Ubuntu server also needs security patches. These are separate from OpenClaw and protect the underlying computer.

To check for security updates:

```
sudo apt update && apt list --upgradable | grep security
```

This shows any security patches waiting to be installed.

Again, your assistant can handle this — it will tell you what needs updating and install it when you approve.

The Easy Way: Let Your Assistant Handle It

Instead of manually checking, just tell your assistant:

> "Check daily for OpenClaw updates and system security updates. Let me know when updates are available, and ask me before installing anything."

Your assistant will add this to its daily routine. Each day, it checks both OpenClaw and system security updates. When something is available, it will message you with details and wait for your approval before installing.

The update process looks like this:

1. Your assistant: "A new OpenClaw version is available (v2026.2.1). Changes: improved memory handling, faster search, bug fixes. Should I install it?"
2. You: "Yes, go ahead"
3. Your assistant: "Installing now… Done! I'm restarting to apply the update. Back in a moment!"

That's it! No Terminal commands, no technical knowledge needed — just approve and your assistant does the work.

Important: Your assistant will always ask before installing anything. Updates occasionally require a restart (especially OpenClaw updates), so you'll have a brief moment where your assistant is offline. Your assistant will warn you before this happens.

Other Messaging Channels

OpenClaw can also connect to WhatsApp, Discord, and SMS. Check the OpenClaw documentation when you're ready to explore these.

Browser Automation

Sometimes your assistant needs to interact with websites that don't have APIs — logging into dashboards, filling out forms, or extracting data from web pages. Browser automation gives your assistant the ability to control a web browser, just like you would.

What It Does

Think of it this way: when you visit Amazon's KDP dashboard to check on your book listings, you're clicking around, reading the screen, and making decisions. Browser automation lets your assistant do the same thing — open a webpage, read what's on it, click buttons, fill in forms, and take screenshots.

This is useful for things like: - Logging into services that don't have an API - Filling out web forms automatically - Extracting data from dashboards (sales reports, analytics, etc.) - Taking screenshots of web pages - Monitoring websites for changes

How to Install It

Ask your assistant to install Playwright (the browser automation engine):

> "Install Playwright browser automation with Chromium."

Your assistant will run these commands:

```
npx playwright install --with-deps chromium
```

This installs a lightweight Chromium browser (the engine behind Google Chrome) that your assistant can control. It takes a few minutes and about 500MB of disk space.

Important: If you're running low on disk space, you may want to upgrade your storage to 30GB (from the 20GB we set up earlier) to accommodate the browser engine. You can do this in the AWS Console under EC2 → Volumes → Modify Volume. Your assistant can walk you through it.

How Your Assistant Uses It

Once installed, your assistant can browse the web on your behalf. Just ask naturally:

> "Log into my KDP dashboard and check on my book sales."
>
> "Go to [website] and fill out the contact form with this information…"
>
> "Take a screenshot of my website's homepage so I can see how it looks."

Your assistant will navigate pages, read their content, and interact with them. For sites that require login, you'll need to provide credentials the first time (save them in TOOLS.md so your assistant remembers them).

Some logins require two-factor authentication (2FA) — your assistant will ask you for the code when needed, just like a human assistant would.

The Browser Relay (Advanced)

OpenClaw also has a Chrome extension called the Browser Relay that lets your assistant see and interact with tabs in YOUR browser on your own computer. This is useful when:

- You're on a page and want your assistant to help with what you're looking at
- A site requires your existing login session (cookies, saved passwords)
- You need your assistant to fill in a complex web form you're looking at

To set this up, check the OpenClaw documentation for the Browser Relay extension. It involves installing a Chrome extension and connecting it to your OpenClaw server.

For most people, the server-side Playwright browser (installed above) covers 90% of use cases. The Browser Relay is a nice-to-have for power users.

Skills, Habits, and Knowing Which to Use

As you use your assistant more, you'll develop processes you want done the same way every time. There are two good ways to lock these down, and knowing which to use matters:

Skills are best for complex, multi-step processes where consistency is critical. Think of them as detailed recipe cards your assistant follows precisely. For example, compiling a monthly newsletter involves selecting content, generating a writing tip, rendering an email template, sending you a preview, and archiving used items. That's a lot of steps with specific rules — perfect for a skill. Your assistant can create skills for you:

> "Let's create a skill for our newsletter process. It should document every step so it's done the same way each month."

Behavioral prompts in AGENTS.md are better for ongoing habits that should happen throughout the day without being told. For example, maintaining a daily status report works best as a standing instruction: "After completing any significant task, append it to today's status report immediately." Your assistant reads AGENTS.md every session, so these become automatic habits rather than one-time procedures.

A good rule of thumb:

- If it's a process with a clear start and end (compile a newsletter, process a batch of data, update a document) → **make it a skill**
- If it's an ongoing behavior or daily habit (keep a status report, check email, monitor something) → **add it to AGENTS.md** with a nightly organization task

Both approaches prevent "drift" — where your assistant gradually starts doing things differently over time. The key insight is: **write it down, don't rely on memory.** Your assistant's memory can be compacted or lost between sessions, but skills and AGENTS.md persist forever.

Browse the OpenClaw documentation or ask your assistant to help you identify which of your processes would benefit from being captured as skills.

Conclusion

You did it! You now have your own personal AI assistant running in the cloud, customized to your personality and needs.

Remember: - Your assistant learns about you over time; the more you use it, the better it gets. - Use the REMEMBER convention to teach it important facts. - Need help? Ask your assistant for help (seriously, it can troubleshoot itself!)

Welcome to the world of personal AI assistants. Enjoy! 🚀

Really Technical Stuff (You Probably Don't Need This)

This section covers common issues and how to fix them. Most people will never need this, but it's here just in case.

Issue 1: Assistant Not Responding

Problem: You send messages in Telegram, but your assistant doesn't reply.

Possible causes: - OpenClaw stopped running. - Your server is offline. - You ran out of API credits.

How to fix: 1. **Check AWS Console:** Go to EC2 → Instances. Is your instance "Running"? If not, select it and click **Start instance**. 2. **SSH into your server:** Check OpenClaw status. `openclaw gateway status` If it says "stopped" or "inactive", restart it: `openclaw gateway restart` 3. **Check API credits:** Log into console.anthropic.com. Add more credits if needed.

Issue 2: Messages Delayed or Slow

Problem: Your assistant responds, but it takes a long time.

Possible causes: - Complex request (web search, browser automation). - Server overloaded. - Network latency.

How to fix: This is usually normal for complex tasks. If *everything* is slow, check your server's resources (CPU/Memory) via SSH. If usage is constantly at 100%, consider upgrading your instance size.

Issue 3: API Key Errors

Problem: Your assistant says "API key invalid" or "API authentication failed."

How to fix: Your key might have expired. Get a new one from console.anthropic.com, then tell your assistant: > "Update my Anthropic API key: sk-ant-..."

Issue 4: Out of Disk Space

Problem: Your assistant says "No space left on device" or operations fail mysteriously.

How to fix: SSH into your server and check disk usage (`df -h`). If usage is above 90%: 1. **Clean up:** Ask your assistant to "Clean up old logs and temporary files." 2. **Expand storage:** In AWS Console (EC2 → Volumes → Modify Volume), increase the size, then tell your assistant to expand the filesystem.

Issue 5: Server Can't Be Reached via SSH

Problem: Connection refused or timed out.

Possible causes: - Server stopped. - IP address changed (if you rebooted). - Security group rules (port 22). - Key permissions.

How to fix: 1. **Check AWS:** Is the instance running? 2. **Check IP:** Did the Public IPv4 address change? Use the new one. 3. **Check Security Group:** Ensure port 22 is open to your IP or "Anywhere". 4. **Check Key Permissions:** Make sure `chmod 400 your-key.pem` was run (Mac/Linux).

Best Practices to Avoid Issues

- Keep OpenClaw and your system updated.
- Monitor costs and disk space regularly.

- Set up automatic backups.
- Don’t install random software from untrusted sources.

Webhooks

External services can trigger your assistant automatically. For example: - Teachable can alert you when students enroll. - Stripe can notify you when payments come through. - Your website contact form can forward inquiries.

See the OpenClaw documentation at docs.openclaw.ai for setup details. Just ask your assistant to help you set these up. ### Real-Time Email Monitoring

By default, your assistant can check email via scheduled cron jobs (e.g., every hour, or twice daily). That works fine for most people, but what if you want your assistant to respond to your emails instantly?

Enter **IMAP IDLE** — a way for the email server to tap your assistant on the shoulder the instant a new email arrives, instead of your assistant having to keep checking. Think of it like this: instead of walking to your mailbox every half hour to see if you have mail, the mail carrier rings your doorbell the moment a letter arrives.

Why You’d Want This

- **Instant responses:** Your assistant sees your email within seconds and can reply or take action immediately
- **No waiting:** Instead of “I’ll check on the next scheduled cron run,” your assistant is already on it
- **More natural:** Email conversations feel more like real conversations

How It Works

A Python script maintains a persistent connection to Gmail using IMAP IDLE. When a new email arrives from your address, the email server notifies the script immediately, and the script triggers an OpenClaw wake event — basically telling your assistant “Hey, you have mail from the owner, go check it now!”

The script runs as a systemd service, which means it starts automatically when your server boots and restarts itself if it ever crashes.

Prerequisites

You need a Gmail account for your assistant with an app password set up. We covered this in **Part F: Setting Up a Dedicated Gmail Account** — if you haven’t done that yet, go back and set it up first.

The Script

Here’s the complete, working email watcher script. Your assistant can help you set this up, but if you’re doing it yourself, save this as `email-watcher.py` somewhere in your workspace (like `~/clawd/scripts/`):

```
#!/usr/bin/env python3
"""
IMAP IDLE Email Watcher
Monitors a Gmail inbox and triggers OpenClaw wake events for emails from the
owner.
"""

import imaplib
import email
import select
import socket
import time
import sys
import re
import subprocess
from datetime import datetime

# ===== CONFIGURATION =====
# Your assistant's Gmail address and app password
IMAP_HOST = "imap.gmail.com"
IMAP_USER = "your-assistant@gmail.com"
IMAP_PASSWORD = "xxxx xxxx xxxx xxxx"  # Gmail app password

# Your email addresses (the assistant watches for emails from you)
OWNER_ADDRESSES = [
    "you@example.com",
    "you@work.com"
]

# Path to openclaw (run 'which openclaw' to find yours)
OPENCLAW_PATH = "/usr/local/bin/openclaw"

IDLE_TIMEOUT = 1740  # 29 minutes (Gmail drops IDLE connections after ~29
min)
MAX_BACKOFF = 300    # Max reconnection delay (5 minutes)
# ========================

def log(message):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    print(f"[{timestamp}] {message}", flush=True)

def extract_email_address(from_header):
    if not from_header:
        return None
    match = re.search(r'<([^>]+)>|([^\s<>]+@[^\s<>]+)', from_header)
    if match:
        return (match.group(1) or match.group(2)).lower()
    return None
```

```
def is_from_owner(from_address):
    if not from_address:
        return False
    return any(addr.lower() in from_address.lower() for addr in
OWNER_ADDRESSES)

def trigger_wake_event(subject):
    try:
        message = f"New email from owner: {subject}. Check inbox and
respond."
        cmd = [OPENCLAW_PATH, "system", "event", "--mode", "now", "--text",
message]
        log(f"Triggering wake event: {message}")
        result = subprocess.run(cmd, capture_output=True, text=True,
timeout=10)
        if result.returncode == 0:
            log("Wake event triggered successfully")
        else:
            log(f"Wake event failed: {result.stderr}")
    except Exception as e:
        log(f"Error triggering wake event: {e}")

def fetch_email_details(mail, email_id):
    try:
        status, data = mail.fetch(email_id, '(RFC822.HEADER)')
        if status != 'OK':
            return None, None
        msg = email.message_from_bytes(data[0][1])
        return msg.get('From', ''), msg.get('Subject', '(no subject)')
    except Exception as e:
        log(f"Error fetching email details: {e}")
        return None, None

def connect_imap():
    log(f"Connecting to {IMAP_HOST}...")
    mail = imaplib.IMAP4_SSL(IMAP_HOST)
    mail.login(IMAP_USER, IMAP_PASSWORD)
    log("Connected and authenticated successfully")
    return mail

def idle_wait(mail, timeout):
    tag = mail._new_tag().decode()
    mail.send(f'{tag} IDLE\r\n'.encode())

    sock = mail.socket()
    old_timeout = sock.gettimeout()
    sock.settimeout(30)
    try:
        response = mail.readline()
        if b'+' not in response:
            log(f"IDLE not accepted:
{response.decode(errors='replace').strip()}")
            return 'error'
    except Exception as e:
        log(f"Error reading IDLE response: {e}")
        return 'error'
    finally:
        sock.settimeout(None)

    log("IDLE mode active, waiting for new emails...")

    result = 'timeout'
    try:
        readable, _, _ = select.select([sock], [], [], timeout)
        if readable:
            sock.settimeout(5)
            try:
                data = sock.recv(4096)
                decoded = data.decode(errors='replace')
                log(f"IDLE notification: {decoded.strip()}")
                if 'EXISTS' in decoded:
                    result = 'new_mail'
            except Exception as e:
                log(f"Error reading IDLE data: {e}")
                result = 'error'
        else:
            log("IDLE timeout reached (29 min), re-entering IDLE")
    except Exception as e:
        log(f"Error in select(): {e}")
        result = 'error'

    try:
        sock.settimeout(30)
        mail.send(b'DONE\r\n')
        for _ in range(10):
            resp = mail.readline()
            if resp and tag.encode() in resp:
                break
    except Exception as e:
        log(f"Error exiting IDLE: {e}")
        result = 'error'
    finally:
        sock.settimeout(old_timeout)

    return result

def process_new_emails(mail):
    try:
        mail.select("INBOX")
        status, messages = mail.search(None, 'UNSEEN')
        if status != 'OK' or not messages[0]:
            return
        for email_id in messages[0].split():
            from_header, subject = fetch_email_details(mail, email_id)
            if from_header:
                from_address = extract_email_address(from_header)
                if is_from_owner(from_address):
                    log(f"Email from owner: {subject}")
                    trigger_wake_event(subject)
    except Exception as e:
        log(f"Error processing emails: {e}")

def main():
    log("Email watcher starting...")
    backoff = 1
    while True:
        mail = None
        try:
            mail = connect_imap()
            mail.select("INBOX")
            backoff = 1
            while True:
                result = idle_wait(mail, IDLE_TIMEOUT)
                if result == 'new_mail':
                    process_new_emails(mail)
                elif result == 'error':
                    break
        except Exception as e:
            log(f"Connection error: {e}")
        if mail:
            try: mail.close(); mail.logout()
            except: pass
        log(f"Reconnecting in {backoff}s...")
        time.sleep(backoff)
        backoff = min(backoff * 2, MAX_BACKOFF)

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        log("Stopped")
        sys.exit(0)
```

Important technical note: This script uses `select.select()` for waiting instead of socket timeouts. This matters! Using `socket.setdefaulttimeout()` will cause the IDLE connection to drop after the timeout period instead of waiting the full 29 minutes. If your assistant writes this script from scratch, make sure it uses `select.select()`.

The Systemd Service

To run the script automatically as a background service, create a systemd service file. Save this as `/etc/systemd/system/email-watcher.service`:

```
[Unit]
Description=Email Watcher for OpenClaw
After=network.target

[Service]
Type=simple
User=ubuntu
ExecStart=/home/ubuntu/your-workspace/venv/bin/python3 /home/ubuntu/your-
workspace/scripts/email-watcher.py
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Adjust the paths to match your setup — use `which python3` to find your Python path if you're not using a virtual environment.

Then enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable email-watcher
sudo systemctl start email-watcher
```

Check that it's running:

```
sudo systemctl status email-watcher
```

The Easy Way: Ask Your Assistant

Instead of manually setting all this up, just give your assistant this prompt:

> "I'd like you to set up real-time email monitoring. Here's a Python script that watches my Gmail inbox using IMAP IDLE and triggers a wake event when I send you an email. Please save this as a script, set it up as a systemd service so it runs automatically, and test it by having me send you a test email."

Then paste the script above (with your real credentials filled in), and your assistant will handle the setup, testing, and troubleshooting.

Testing: Once it's running, send a test email to your assistant's Gmail address. You should see your assistant respond within a few seconds instead of waiting for the next scheduled check!

Last updated: February 2026 **OpenClaw version:** 2026.x