Revision 32 — May 6, 2026

So You Want To Set Up Your Own OpenClaw Server

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

This is a quick-reference guide with the minimum steps needed to create an OpenClaw server. For a comprehensive course with video walkthroughs and detailed explanations, check out the full class on Udemy: Easy OpenClaw: Create An Employee Who Works for Free

Table of Contents

1. **Part A: Prerequisites & Accounts** — What you need before starting
2. **Part B: AWS Account & Server Setup** — Creating your cloud computer (includes getting your API key)
3. **Part C: Telegram Setup** — Get this ready before installing OpenClaw
4. **Part D: Installing OpenClaw** — Getting your assistant running
5. **Part E: Your First Hour** — Configuring your assistant through conversation
6. **Part F: Optional API Keys & Services** — Adding extra capabilities
7. **Part G: Getting the Most Out of Your Assistant** — Tips and features
8. **Part H: What Else Can Your Assistant Do?** — Next steps to explore
9. **Conclusion**
10. **Really Technical Stuff** — Troubleshooting (you probably won't need this)
    - Real-Time Email Monitoring

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) - Your server costs a modest monthly fee — less than most streaming subscriptions - API costs (the AI “brain”) are pay-as-you-go depending on how much you chat - The total is comparable to a single AI chat subscription, but you get your own server and professional-level access

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

Let’s talk money because surprises are no fun.

Your server (AWS Lightsail): A flat monthly fee for your cloud computer — simple, predictable, no surprise charges. You pick a plan when you create your server and that’s what you pay. Check current Lightsail pricing at aws.amazon.com/lightsail/pricing.

AI API costs (Anthropic Claude): This is pay-as-you-go based on how much you chat. Light use costs less, heavy use costs more. It scales naturally with how much value you’re getting.

- **Initial deposit: $50 recommended** (ensures you don’t hit rate limits during setup)
- Optional APIs like OpenAI (voice transcription), Brave Search, and Google AI (image generation) have generous free tiers

The bottom line: For roughly what you'd pay for a single AI chat subscription, you get your own always-on server, professional pay-as-you-go AI access, persistent memory, background automation, and complete control. No per-seat licensing, no feature tiers, no waiting for someone else to add the features you want.

Things You'll Save During This Process

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

- ☐ AWS account password
- ☐ Anthropic API key (starts with sk-ant-)
- ☐ Your Telegram User ID (a number)
- ☐ Your Telegram Bot Token (a long string)
- ☐ AWS backup credentials (Access Key ID and Secret Access 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 a support plan — **Basic** (free) is fine for our purposes.
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: Getting Your Anthropic API Key

Your cloud server needs someone smart on the other end to have a conversation with. That's where Anthropic comes in. They make Claude, the AI that will power your assistant's intelligence and personality.

Think of an API key like a special password that lets your assistant talk to Claude's brain.

1. Go to **console.anthropic.com**
2. Enter your email address and click "Continue with email"
3. Check your email for a sign-in link from Claude Console and click it
4. Enter your full name, choose a nickname, agree to the terms, and click Continue
5. Choose **Individual**
6. You're in! This is the Claude Console — your home base for managing your API key and monitoring usage.

Now let's get your API key:

1. Click **API Keys** in the left menu (under "Manage")
2. Click **Create Key**
3. Give it a name like "OpenClaw" so you remember what it's for
4. Leave the workspace on Default and click **Add**
5. You'll see your API key — a long string starting with `sk-ant-`
6. **Click "Copy Key" immediately** and paste it into your notes
7. You won't be able to see this key again after closing this window!

Set up billing so your assistant can use the API:

1. Click **Billing** in the left menu
2. Click **Add credit card** and enter your payment information
3. Once your card is added, click **Edit** next to the auto-reload warning
4. Turn on **Auto-reload** and set whatever amounts you're comfortable with — this way your credits automatically refill when they get low
5. To set a spending limit, click **Limits** and scroll down — you can set email notifications at whatever amount you choose

Important: Start with at least **$50 in credits** and set auto-reload to trigger at **$20**. This isn't just about having enough money — Anthropic's rate limits are directly tied to your credit balance. A low balance means you'll hit "cooldown" warnings where your assistant temporarily can't respond, which is frustrating during setup when you're chatting frequently. A $50 balance puts you in a comfortable rate limit tier where you won't run into this. Typical monthly usage is $15-50 depending on how much you chat, so $50 lasts most people at least a month.

 Checkpoint: You should now have your Anthropic API key saved (starts with `sk-ant-`). Keep it handy — we'll need it when we install OpenClaw in Part D.

Step 3: Launching Your Server (Lightsail Instance)

Okay, here's where we actually create your cloud computer! AWS Lightsail is Amazon's simplified server service — think of it as "renting a computer that stays on all the time" with a flat monthly price and no surprises.

Finding the right place:

1. In the AWS Console, click on the search bar at the top and type **Lightsail**.
2. Click on **Lightsail** in the search results.
3. If this is your first time, you'll see a welcome page. Click **Let's get started** or **Create instance**.

Creating your instance:

1. **Region:** Choose a region close to you. The default is usually fine.
2. **Platform:** Select **Linux/Unix**.
3. **Blueprint:** Under **OS Only**, select **Ubuntu** (the latest LTS version).
4. **Instance plan:** Choose the **4 GB RAM** plan. This gives you plenty of room for OpenClaw to run comfortably, with 80 GB of storage included.
5. **Name:** Give it a name like "My-OpenClaw-Server".
6. Click **Create instance**.

Your instance will appear in the list. Wait for it to show **Running**. That means your server is booting up!

Assign a Static IP:

By default, your server's IP address can change if it restarts. Let's fix that:

1. Click on your instance name to open its details.
2. Go to the **Networking** tab.
3. Under **IPv4 networking**, click **Attach static IP**.
4. Give it a name like "OpenClaw-IP" and click **Create and attach**.
5. **Write down this IP address** — it's permanent and you'll need it to connect to your server.

Step 4: Connecting to Your Server

Lightsail makes this incredibly simple — no special key files, no Terminal setup. You connect right from your web browser.

1. On the Lightsail home page, find your instance.
2. Click the little **terminal icon** (▯) on your instance, or click the instance name and then click **Connect using SSH**.
3. A black terminal window opens right in your browser!

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

That's it — you're connected! No software to install, no key files to manage.

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

Optional: Connecting from your own Terminal

If you prefer using your computer's Terminal (Mac) or PowerShell (Windows) instead of the browser, you can download an SSH key from Lightsail:

1. Go to your Lightsail **Account** page (click your username at the top right).
2. Click the **SSH keys** tab.
3. Click **Download** next to the default key for your region.
4. Save the `.pem` file somewhere safe.
5. Mac users: Run `chmod 400 ~/Downloads/your-key-name.pem` to lock down the file.
6. Connect: `ssh -i ~/Downloads/your-key-name.pem ubuntu@YOUR-IP`

Most people find the browser console easier, but the option is there if you want it.

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 `AdministratorAccess`. Select the checkbox for **AdministratorAccess**. Click **Next**. *(Note: AWS removed the Lightsail-specific policy in early 2026. AdministratorAccess is broader than necessary, but it works reliably for everything in this guide.)*
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 browser 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 browser console open from Part B. If so, you’ll see that `ubuntu@ip-something:~$` prompt and you’re ready to go.

If you closed it, no worries — just go back to **lightsail.aws.amazon.com**, find your instance, and click the **terminal icon** () or click the instance name and then **Connect using SSH**. A browser console opens right up.

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 -s -- --version
2026.5.5
```

That’s it. This single command downloads the OpenClaw installer, which then installs everything your server needs — Node.js, Git, and OpenClaw itself. We’re specifying version 2026.5.5, which is the current stable release (May 2026). It includes months of bug fixes for plugin handling, streaming display, gateway reliability, and provider support. We pin to a specific version so your install matches this guide exactly.

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 Anthropic key you saved in Part B (starts 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 browser console** where OpenClaw is running. Type this command:

```
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 browser console

Go ahead and close your browser console. 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: Your First Hour

Now for the fun part — making this assistant actually yours!

Your assistant uses text files behind the scenes to remember who you are and how to behave. But you don’t need to edit any files yourself. Just talk to it. Work through these messages one at a time — copy, paste, and send each one via Telegram. Each builds on the last.

Most of these take less than a minute. The whole list takes about 30 minutes, and you’ll have a well-configured, secure, personalized assistant when you’re done.

1. Give It a Name and Personality

Your assistant starts generic. Fix that first.

> “Let’s set up your personality. Your name is [pick a name you like]. Write a SOUL.md file that describes who you are — your name, your vibe, how you communicate. Be direct and helpful, not corporate or

> sycophantic. Have opinions. Be concise when the situation calls for it and thorough when it matters."

What this does: Creates SOUL.md, which shapes how your assistant talks and behaves across every conversation. It persists through restarts.

2. Tell It About You

The more context your assistant has, the better it can help.

> "Let's create my USER.md profile. My name is [your name]. I live in [city/timezone]. Here's what I do for work: [brief description]. My communication style preference: [e.g., 'be direct, skip the fluff' or 'explain things thoroughly']. I mainly use you via Telegram on my [phone/computer/both]."

What this does: Creates USER.md so your assistant knows your timezone, work context, and preferences. Add more details anytime by saying "Update USER.md with..."

3. Set Your First REMEMBER Rules

Whenever you say "REMEMBER" (in all caps) followed by information, your assistant immediately saves it to long-term memory. This works in every conversation, forever. It's one of the most powerful features you have. Here are three good ones to start with:

> "REMEMBER: Always present times in my local timezone, not UTC."
>
> "REMEMBER: When I ask you to do something and you're unsure, ask me a clarifying question rather than guessing."
>
> "REMEMBER: If you're not sure about something, say so. Don't make things up."

Keeping Memory Fresh Over Time: As your assistant's memory grows over weeks and months, older notes can crowd out newer, more relevant ones when it searches for information. OpenClaw has a built-in feature called *temporal decay* that automatically gives newer memories higher priority. To turn it on, just say: "Enable temporal decay for memory search with a 30-day half-life." Older memories don't get deleted — they just carry less weight, so your assistant naturally prioritizes recent information.

4. Secure Your Server

These three messages harden your setup against attacks.

> "Install fail2ban and configure it to ban IP addresses after 3 failed SSH login attempts for 24 hours."

> “Audit your own security setup and tell me if anything needs attention. Check SSH configuration, open ports, system updates, and anything else that might be a vulnerability.”
>
> “REMEMBER: Run a security self-audit once a month and let me know if anything needs attention.”

What these do: Fail2ban blocks brute-force attacks automatically. The audit checks your current security posture. The REMEMBER ensures ongoing vigilance.

5. Set Up Monitoring and Alerts

Your assistant can watch itself and only bother you when something’s actually wrong.

> “Set up monitoring to alert me if: you stop running and can’t respond, the server disk space gets above 80%, or there’s a system security update available. Check these daily and only message me when something needs my attention.”

What this does: Creates background scheduled tasks (cron jobs) that run quietly. You’ll hear nothing most days — which means everything is fine.

6. Set Up Automatic Updates

Keep everything current without thinking about it.

> “Check for OpenClaw updates and system security updates every day. When an OpenClaw update is available, tell me what’s new and ask before installing. For security updates, install them automatically and just let me know what was patched.”

What this does: Your assistant monitors for both OpenClaw feature updates (asks before installing) and OS security patches (installs automatically). You stay current without manual checking.

7. Verify Your Backups Are Working

You set up backup credentials in Part B. Now make sure backups are actually running.

> “Check my backup setup. When was the last backup taken? Is the schedule running correctly?”

What this does: Confirms your safety net is in place. If something ever goes wrong, you can restore from a backup.

8. Set a Cost Warning

Avoid surprise bills.

> “REMEMBER: Monitor our conversation length. When context is getting high, warn me so I can decide whether to compact the conversation or keep going. Long conversations cost more because you re-read the entire history with each message.”

Then check your current usage:

> /status

What this does: The REMEMBER sets a permanent alert threshold. The /status command shows your current context usage anytime. Use /compact to shrink a long conversation, or /new to start completely fresh.

9. Teach It the Recovery Procedure

In case you ever need to troubleshoot.

> “REMEMBER: If I ever say ‘you were unresponsive’ or ‘Telegram stopped working,’ walk me through the recovery steps: go to lightsail.aws.amazon.com, click my instance, click Connect using SSH to open the browser console, then type ‘openclaw tui’ to verify you’re running. If the TUI works but Telegram doesn’t, tell me to run ‘openclaw gateway restart’.”

What this does: Your assistant will know how to guide you through recovery even if you forget the steps.

10. Set Up Background Tasks

Your assistant can run tasks on a schedule without you lifting a finger. These are called **cron jobs** — they run in isolated sessions, separate from your conversation, and only message you when something needs your attention.

Steps 5 and 6 above already created your first cron jobs (monitoring and updates). Here are more ideas:

> “Check my email every morning at 7 AM and send me a digest of anything important.”
>
> “Remind me every Sunday evening to plan my week.”

You can always ask “What cron jobs do I have running?” to see your full automation schedule. Any recurring task you can describe, your assistant can schedule.

11. Take It for a Spin

Now that everything is configured, try some real tasks to see what your assistant can do.

> "Search the web for [something you're interested in] and give me a summary."
>
> "What day of the week is my birthday, [date]? Add it to your memory so you remember next year."
>
> "Write a short thank-you note to [person] for [reason]. Just show me the text."
>
> "What files are in my workspace? Give me an overview of how things are organized."

What these do: Gets you comfortable with your assistant's capabilities — web search, memory, writing, and file management. Try anything you'd ask a human assistant to do.

What's Next?

You now have a personalized, secured, monitored assistant that knows who you are and how you like to work. Everything you just did — the personality, your profile, the REMEMBER rules, the security, the monitoring — your assistant saved all of it into files that persist across every conversation. These are living documents. You can update them anytime just by talking:

- "I'd like you to be a bit more casual — update your personality"
- "Add to my profile that I'm learning Spanish"
- "REMEMBER my anniversary is June 15th"

From here:

- **Explore Part F** to add optional capabilities (image generation, voice transcription, web search API)
- **Ask your assistant to create skills** for any multi-step process you do repeatedly
- **Just use it** — the best way to learn what it can do is to ask it to do things

If something goes wrong, remember: Lightsail Console → click your instance → Connect using SSH → type `openclaw tui`.

Part F: Optional API Keys & Services

You already set up the most important API key — Anthropic (Claude) — back in Part B. Everything below is **optional** and adds extra capabilities to your assistant. You can set these up now or come back to them later.

Good news: Just tell your assistant your API keys and it will add them to the configuration for you!

1. 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.

2. 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.

3. 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!

4. X (Twitter) Pro Subscription (For Sharing Articles)

What it is: A $8/month X subscription that includes API access.

Why you want it: One of the most productive daily workflows is sharing interesting articles, threads, and posts from X directly with your assistant. You forward a link; your assistant reads the full text, summarizes it, and saves the useful parts. Without Pro, X blocks all automated access — your assistant can’t read any X content you share.

How to get it:

1. Go to x.com and sign up for X Pro ($8/month)
2. That’s it — the API access is included automatically

Your assistant already knows how to read X posts using the API. Once you have Pro, just paste any X link into your conversation and ask about it.

Cost: $8/month. Pays for itself quickly if you regularly share articles and threads with your assistant.

Setting Up Cloud 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 cloud storage 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.

Which should you pick?

- **Google Drive** (recommended if you use Gmail) — Easier to set up, and once configured it never expires. No tokens to refresh, no re-authorization months later.
- **Dropbox** — Good alternative if you already use Dropbox and prefer to keep everything there.

Pick whichever service you already use. You only need one.

Option A: Google Shared Drive (Recommended for Gmail Users)

This creates a shared space on Google Drive where you and your assistant can exchange files. Once set up, it works forever — no tokens to refresh.

Step 1: Create a Google Cloud Project

1. Go to **console.cloud.google.com**
2. Sign in with your Google or Gmail account
3. Click the project dropdown at the top of the page (it may say “Select a project”)
4. Click **New Project**
5. Name it something like “My Assistant” and click **Create**
6. Make sure your new project is selected in the dropdown at the top

Step 2: Enable the Google Drive API

1. In the left sidebar, click the hamburger menu (three horizontal lines) and go to **APIs & Services** then **Library**
2. Search for “Google Drive API”
3. Click on it, then click **Enable**

Step 3: Create a Service Account

A service account is like a special Google account just for your assistant. It can read and write files, but it can’t log in or browse the web.

1. In the left sidebar, go to **APIs & Services** then **Credentials**
2. Click **Create Credentials** and choose **Service Account**
3. Name it something like “openclaw-bot” and click **Create and Continue**
4. You can skip the optional role and user access steps — just click **Done**
5. You’ll see your new service account listed. Click on its name.
6. Go to the **Keys** tab
7. Click **Add Key**, then **Create new key**, choose **JSON**, and click **Create**
8. A JSON file will download to your computer. **Keep this file — you’ll need it in Step 5.**

Step 4: Create a Shared Drive

1. Go to **drive.google.com**
2. In the left sidebar, look for **Shared drives** (you may need to scroll down)
3. Click the **+** button or **New shared drive**
4. Name it something like “Assistant Files”
5. Click **Create**

Step 5: Share the Drive with Your Assistant

1. Right-click your new shared drive and choose **Manage members**
2. Open the JSON file you downloaded in Step 3. Look for the line that says `"client_email"` — it will look something like `openclaw-bot@my-assistant-12345.iam.gserviceaccount.com`
3. Paste that email address into the “Add people” field
4. Set the role to **Content Manager**
5. Click **Send** (or **Share**)

Step 6: Tell Your Assistant

Send your assistant a message like:

> “I’ve set up Google Drive for file sharing. Here’s my service account key.”

Then paste the contents of the JSON file, or send it as an attachment. Your assistant will install the necessary tools and configure everything from there.

If you run into trouble at any step, tell your assistant what you’re seeing and it can help you through it.

Option B: Dropbox

If you prefer Dropbox, you can connect your assistant to any Dropbox account — including a team or family account.

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 using the Dropbox Python SDK (not rclone) and save files to the Assistant/ folder.”

Your assistant will create the necessary script and configuration automatically!

How to Use Cloud File Sharing

Once either option is set up, you can say things like:

- “Save that report to my cloud folder”
- “I put a photo in the shared drive — can you look at it?”
- “Generate an image and save it for me”

Your assistant will handle the rest. Files you put in the shared folder appear for your assistant automatically, and files your assistant saves there appear on your computer.

Tip: Create subfolders to stay organized (e.g., Images, Documents, 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.

Consolidate tasks to save money: Every time a cron job runs, your assistant has to reload its workspace files — personality, memory, instructions — before it can do anything. This loading step (called a "cache write") costs the same whether the task takes 5 seconds or 5 minutes. Ten separate cron jobs running at 10 different times cost roughly 10 times as much as one combined job that does all 10 things in a single run.

The fix is simple: batch routine tasks together. Instead of separate cron jobs for checking email, running reports, monitoring server health, and syncing files, create one daily "housekeeping" job that does all of them in order. Content-generation tasks (writing social media posts, compiling newsletters) still benefit from their own runs, but anything that just runs a script and reports the results should be combined.

As a real-world example: one server had an email checker running every 30 minutes and a newsletter monitor running every 15 minutes — 144 runs per day just for those two jobs. Consolidating everything into a single daily batch cut the AI costs by more than half.

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

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.

Longer conversations cost more because there’s more to re-read with each message. When the context window fills up, your assistant automatically compacts the conversation — summarizing older material to make room. This works well, but some details can get lost in the summary. That’s why shorter, focused conversations tend to produce better results and cost less.

Where Your Money Actually Goes

Your Lightsail server is a flat monthly fee — predictable and simple. The variable cost is AI API usage.

Most of your API cost is *input*, not output. Every time you send a message, your assistant re-reads its entire context — system instructions, workspace files, tool definitions, and the full conversation history. For a typical setup, that’s 30,000–50,000 tokens *before it even starts thinking about your question*.

This is why long conversations get expensive, and why the tools below save real money. They reset the conversation history so your assistant isn’t re-reading an hour of old messages every time you ask something new. The workspace files still get loaded each time (that’s how it remembers who it is), but keeping those files lean means that baseline cost stays low.

How Caching Saves You Money (and When It Stops)

Anthropic keeps your conversation cached on their servers for **five minutes** after each message. If your next message arrives within that window, the cached portion costs about one-tenth the normal rate. Only the new content — your latest message and the assistant’s reply — gets charged at full price.

This means a rapid back-and-forth conversation stays cheap even as the context grows, because most of it is cached. But if you step away for more than five minutes — check your email, take a call, grab lunch — the cache expires. The next message reloads the *entire* conversation at full price.

Early in a session, a cache miss barely matters. At 30,000 tokens of context, reloading costs a few cents. But at 500,000 tokens — after hours or days of accumulated conversation — a single cache miss can cost $10 or more. And if you're sending messages with gaps longer than five minutes at that context size, *every turn* hits you at full price.

This is the hidden trap: a long session isn't expensive because it's long. It's expensive when it's **long and intermittent** — big context with gaps between messages. The fix is simple: when you're done with a topic or coming back after a break, use `/new` to start fresh. Your assistant's permanent memory carries over, the context drops back to 30,000–50,000 tokens, and cache misses cost pennies again.

When your assistant warns you that context is getting 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.

Watch Your Workspace File Sizes

Your workspace files (SOUL.md, MEMORY.md, PLAYBOOK.md, etc.) are loaded at the start of every message. But each file has a size limit — 20,000 characters by default. If a file exceeds that, OpenClaw silently trims the middle, keeping the beginning and end but cutting content from the center. No error, no warning. Your assistant thinks it read the whole file. You think it read the whole file. But rules buried in the middle may have been cut.

To check, type `/context list` in chat. If you see **TRUNCATED** next to a file, it's been trimmed. The fix is simple: split large files into smaller, focused ones — which is better organization anyway. (Lesson 5.1 in the course covers a good strategy for this.) If you really need a bigger file, tell your assistant: "Set bootstrapMaxChars to 40000."

Daily Session Reset: By default, OpenClaw resets your session once a day (at 4:00 AM on the server's clock). This means you start each morning with a fresh context window — no need to manually compact. Your assistant's

permanent memory (MEMORY.md, PLAYBOOK.md, etc.) carries over automatically, so nothing important is lost. You can change the reset time by asking your assistant: “Set the daily session reset to 4 AM Eastern time” (or whatever time works for you — ideally a time when you’re never actively chatting).

Advanced Tip: Event-Driven Alerts from Background Jobs

Here’s a powerful pattern once you’re comfortable with cron jobs: your background automation can “wake up” your main conversation when something important happens.

For example, suppose you have a cron job that checks your email every 30 minutes. Normally, it runs in its own isolated session and sends you a Telegram alert if something needs attention. But when you reply to that alert, your assistant in the main conversation doesn’t know what you’re referring to — because the email check happened in a separate session.

The fix: tell your assistant to set up its email monitoring so that when it finds something important, it also injects a summary into the main session using a one-shot system event. That way, when you reply about the email, your assistant already has the context.

This is the kind of refinement you’ll discover naturally as you use your assistant. Just describe what’s frustrating (“when I reply about an email alert, you don’t know what I’m talking about”) and your assistant will figure out the right architecture.

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!

Keeping Conversations Separate: By default, all users share the same conversation session. If you'd prefer each person to have their own private conversation with your assistant (recommended when adding family members), ask your assistant: "Set up per-user sessions so each person has their own conversation." This uses a setting called `dmScope: per-channel-peer` — your assistant knows how to configure it. Each person gets their own session with separate context, while sharing the same assistant personality and memory files.

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.

Quick Config Backup: In addition to full server snapshots, OpenClaw has its own backup command that saves your configuration, credentials, and workspace files into a single archive:

```
openclaw backup create
```

This is faster than a full server snapshot and captures everything that makes your assistant *yours* — personality, memory, rules, API keys, all of it.

If you ever need to start over on a new server, this archive gets you back to exactly where you were. Tell your assistant to run this weekly alongside your AWS snapshots — belt and suspenders.

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: Do NOT auto-update OpenClaw.** New OpenClaw releases frequently introduce bugs — memory leaks, plugin crashes, broken features. Your server's auto-update has been disabled for this reason. When your assistant reports a new version is available, **wait at least 48 hours**, then ask your assistant to check Reddit (r/openclaw) and GitHub issues for complaints about the new version. Only update when you're confident the release is stable. When in doubt, ask Steve. Your assistant will always ask before installing anything, and updates require a brief restart where your assistant is temporarily offline.

Keeping Your Server Secure

Your AWS server is already more secure than most setups out of the box. Let's make sure you understand why, and add a few easy improvements.

What AWS Already Does for You

When you set up your server in Part B, AWS automatically gave you several security features:

SSH Key Authentication: Lightsail uses SSH keys to control access to your server. There's no password to guess — if someone doesn't have the right key, they can't get in. This is much more secure than a username and password.

Lightsail Firewall: Lightsail has a built-in firewall that controls which network traffic can reach your server. By default, only SSH (port 22) and HTTP (port 80) are open — everything else is blocked. Your assistant communicates through Telegram's cloud servers, so it doesn't need any additional ports open to the internet.

Non-Root User: You're logging in as `ubuntu`, not as `root` (the all-powerful administrator account). This means even if someone somehow got access, they couldn't immediately do the most dangerous things without an extra step.

Isolated Server: OpenClaw runs on its own dedicated server in the cloud, completely separate from your personal computer. If anything ever went wrong, your personal files, photos, and data are untouched.

Easy Security Improvements

Here are three things you can do in about five minutes that significantly improve your security:

1. Install Fail2ban (Stops Brute Force Attacks)

Fail2ban watches for repeated failed login attempts and automatically bans the attacker's IP address. Think of it as a bouncer who kicks out anyone who keeps trying wrong passwords.

Tell your assistant:

> "Install fail2ban and configure it to ban IPs after 3 failed SSH attempts for 24 hours."

Your assistant will install and configure it. Once running, it works silently in the background — you'll never need to think about it again.

2. Ask Your Assistant to Audit Its Own Security

This is one of the coolest things about having an AI assistant — it can check its own security! Just ask:

> "Audit your own security setup and tell me if anything needs attention."

Your assistant will check things like: whether SSH is properly configured, whether unnecessary ports are open, whether your system is up to date, and whether there are any obvious vulnerabilities. It's like having a security consultant on call 24/7.

3. Set Up Real-Time Alerts

Your assistant can monitor itself and message you when something is wrong:

> “Set up monitoring to alert me if: the server is running low on disk space, the server hasn’t been backed up recently, OpenClaw stops running, or anything unusual happens.”

Your assistant will create background checks (cron jobs) that run periodically and only message you when something actually needs your attention. Most days, you’ll hear nothing — which means everything is fine.

For Advanced Users

If you want to go further, here are some additional hardening options. These aren’t necessary for most people, but they’re available:

- **Tailscale VPN:** Creates a private network between your devices. Useful if you want to access your server’s web interface directly without exposing any ports to the internet. Free for personal use. See tailscale.com.
- **UFW Firewall:** An additional firewall layer on the server itself, on top of the Lightsail firewall. Belt-and-suspenders approach. **Important:** Do NOT use UFW to restrict SSH access to specific IP addresses. The Lightsail browser console connects from rotating proxy IPs that cannot be reliably whitelisted, and you will lock yourself out of your own server. If you use UFW, keep SSH open to all and rely on key-only authentication + Fail2ban for security.
- **Docker Sandboxing:** Runs sub-agent tasks in isolated containers so they can’t access sensitive files. Significant setup effort, but worth it for high-security environments.

For most users, the combination of the Lightsail firewall + SSH keys + Fail2ban + your assistant’s self-monitoring covers everything you need.

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 (Advanced — Optional)

OpenClaw has the ability to control a web browser on the server using a tool called Playwright. This lets your assistant log into websites, fill out forms, and take screenshots of web pages.

Most people won’t need this. Your assistant can already search the web, fetch web pages, and interact with services that have APIs — all without a browser. Browser automation is mainly useful for sites that require a visual login session or don’t have any other way to access their data.

If you’re interested, just tell your assistant:

> “Install Playwright browser automation with Chromium.”

It takes a few minutes and about 500MB of disk space. But honestly, start without it and add it later only if you find a specific need.

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.

Understanding the Architecture: Workspace Files, Skills, and Sub-Agents

Now that you've been using your assistant for a while, it helps to understand how its memory actually works — because the architecture affects what you should put where.

Workspace Files

Your assistant loads several markdown files from its workspace every time a new session starts:

- **AGENTS.md** — session startup instructions and behavioral guidelines
- **SOUL.md** — personality and voice
- **USER.md** — information about you
- **TOOLS.md** — API credentials and service access
- **MEMORY.md** — persistent facts, project state, people, infrastructure
- **IDENTITY.md**, **HEARTBEAT.md** — identity and background task configuration

These files are injected into the AI's context window at the start of every conversation. That's what makes your assistant "remember" things between sessions; it re-reads these files each time.

The critical limit: Each workspace file is truncated at **20,000 characters**. If MEMORY.md grows past 20K, the end gets silently cut off. Your assistant won't see the missing content and won't know it's missing. The total across all workspace files is capped at **150,000 characters**.

This means you need to keep these files lean. Think of them as expensive real estate; every line competes for space. If a section of MEMORY.md contains detailed procedures you only need occasionally, it's taking up space that could hold facts you need every session.

What Are Skills, Really?

Skills look like instruction files, and they are — but they work differently from workspace files in two important ways:

1. **Skills load on demand.** Your assistant reads a skill's description (a few sentences) at the start of every session, but only loads the full skill when it's relevant to what you're asking. A skill about newsletter compilation doesn't use any context space when you're editing a website.

2. **Skills are visible to sub-agents.** This is the big one. When your assistant spawns a cheaper sub-agent to handle a routine task (like posting to Facebook or processing email), that sub-agent can see and use skills. It cannot see your workspace files like MEMORY.md or custom files you've created.

This means: if you put your Facebook posting rules in MEMORY.md, only your main assistant knows them. The sub-agent actually doing the posting is flying blind. But if those rules are in a skill, the sub-agent has full access.

What Should Go Where?

Workspace files (MEMORY.md, AGENTS.md, any custom .md files):

- Facts that matter every session (people, projects, infrastructure)
- Behavioral rules (how should the assistant act in general)
- Safety rules and hard constraints
- Keep these under ~16,000 characters each to stay safely below the 20K truncation limit

Skills:

- Procedures for specific tasks (how to post to Facebook, how to manage email lists)
- Platform-specific knowledge (API patterns, CSS workarounds, formatting rules)
- Anything a sub-agent might need to do
- Reference data that's only relevant during specific tasks (page inventories, keyword lists)

A practical example: Say you have rules about managing an email marketing list — which contacts go on which list, what's safe to do automatically, what requires your approval. If those rules are in MEMORY.md, your main assistant knows them. But if a scheduled task (cron job) tries to process a new subscriber, it spawns a sub-agent that can't see MEMORY.md. If the rules are in a skill called "email-management," the sub-agent reads the skill and follows the same rules you set.

When to Create a Skill

Ask yourself these questions:

- **Does a sub-agent or cron job need this information?** → Skill.
- **Do I repeat this procedure more than once a month?** → Skill.
- **Is this more than 10 lines of procedure?** → Probably a skill.
- **Is this a fact I need every session (a person's name, a project status)?** → Workspace file.
- **Is this a one-line rule ("always ask before sending email")?** → Workspace file.

You can ask your assistant to create skills for you:

> "Let's turn our email handling procedures into a skill. Include the routing rules, the safety constraints, and the scripts we use."

Your assistant will create the skill files, organize them, and they'll be available to every session and sub-agent from that point on.

Checking Your File Sizes

You can ask your assistant to check how close your workspace files are to the limits:

> “How big are my workspace files? Are any approaching the 20K limit?”

If a file is getting large, ask your assistant to identify what can move into a skill. Anything procedural or task-specific is a good candidate.

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 (Most Common!)

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

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

How to fix — the Lightsail Console method (no Terminal required!):

This is the easiest way to check on your server and fix most problems. You don’t need Terminal, SSH, or any special software — just a web browser.

1. Go to **lightsail.aws.amazon.com** and sign in to your AWS account.
2. You should see your server listed. First, check: is it showing **Running**? If not, click on it and click **Start**, then wait a minute or two.
3. If the instance IS running, click the **terminal icon** () on your instance, or click the instance name and then click **Connect using SSH**.
4. A black terminal window will open right in your browser! You’re now connected to your server.
5. Type: `openclaw tui`

6. If the TUI opens and your assistant responds, try messaging it in Telegram again — it may just need a moment to reconnect.
7. If the TUI shows an error, or if your assistant still isn't responding in Telegram, type: `openclaw gateway restart` Wait about 30 seconds, then try Telegram again.
8. If none of that works, try a full restart: `sudo systemctl restart openclaw-gateway`

Still stuck? Check your API credits at console.anthropic.com. If your balance is zero, your assistant can't think — add more credits and it will come back to life.

Alternative: SSH from Terminal (if you prefer the traditional method)

If you downloaded an SSH key from Lightsail (see Step 4), you can connect from your computer's Terminal:

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

Then run the same commands: `openclaw tui` or `openclaw gateway restart`.

Issue 2: "Cooldown" Warnings or Rate Limiting

Problem: Your assistant says something about "cooldown," "rate limit," or temporarily can't respond. It might suggest adding a different AI provider as a fallback.

What's happening: Anthropic limits how fast you can send messages based on your credit balance. New accounts with small balances ($5-10) have very low limits. During setup, when you're chatting frequently, it's easy to hit them.

How to fix:

1. Go to **console.anthropic.com → Billing**
2. Add credits to bring your balance to at least **$50**
3. Set auto-reload to trigger at **$20** so you never run low

The rate limits increase automatically as your balance grows. With $50 in credits, you'll rarely see cooldown warnings.

⚠ **Important:** If your assistant suggests switching to a different AI provider (like OpenAI) as a fallback, **don't do it** during setup. The assistant behaves differently on other models and may not be able to switch itself back properly. Instead, just wait a minute for the cooldown to pass, or add more credits. If your assistant has already gotten into a confused state from switching providers, the safest fix is to re-run the setup wizard:

```
openclaw setup
```

This regenerates a clean configuration. Walk through the wizard again with your Anthropic key and you'll be back to normal.

Issue 3: 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%, you can upgrade your Lightsail instance: create a snapshot, then create a new instance from that snapshot with a larger plan. Your assistant can walk you through this.

Issue 4: 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 5: 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:** If you need more space, you can upgrade to a larger Lightsail plan by creating a snapshot and launching a new instance from it. Your assistant can walk you through this.

Issue 6: Server Can't Be Reached via SSH

Problem: Connection refused or timed out.

Possible causes: - Server stopped. - Firewall rules blocking SSH. - Key permissions.

How to fix: 1. **Check Lightsail:** Is the instance running? Go to lightsail.aws.amazon.com and check. 2. **Use the browser console:** Click the terminal icon on your instance — this always works even if SSH from your computer doesn't. 3. **Check Lightsail Firewall:** Click your instance → Networking tab → make sure port 22 (SSH) is listed. 4. **Check Key Permissions:** If using Terminal, 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: May 2026 **OpenClaw version:** 2026.5.5