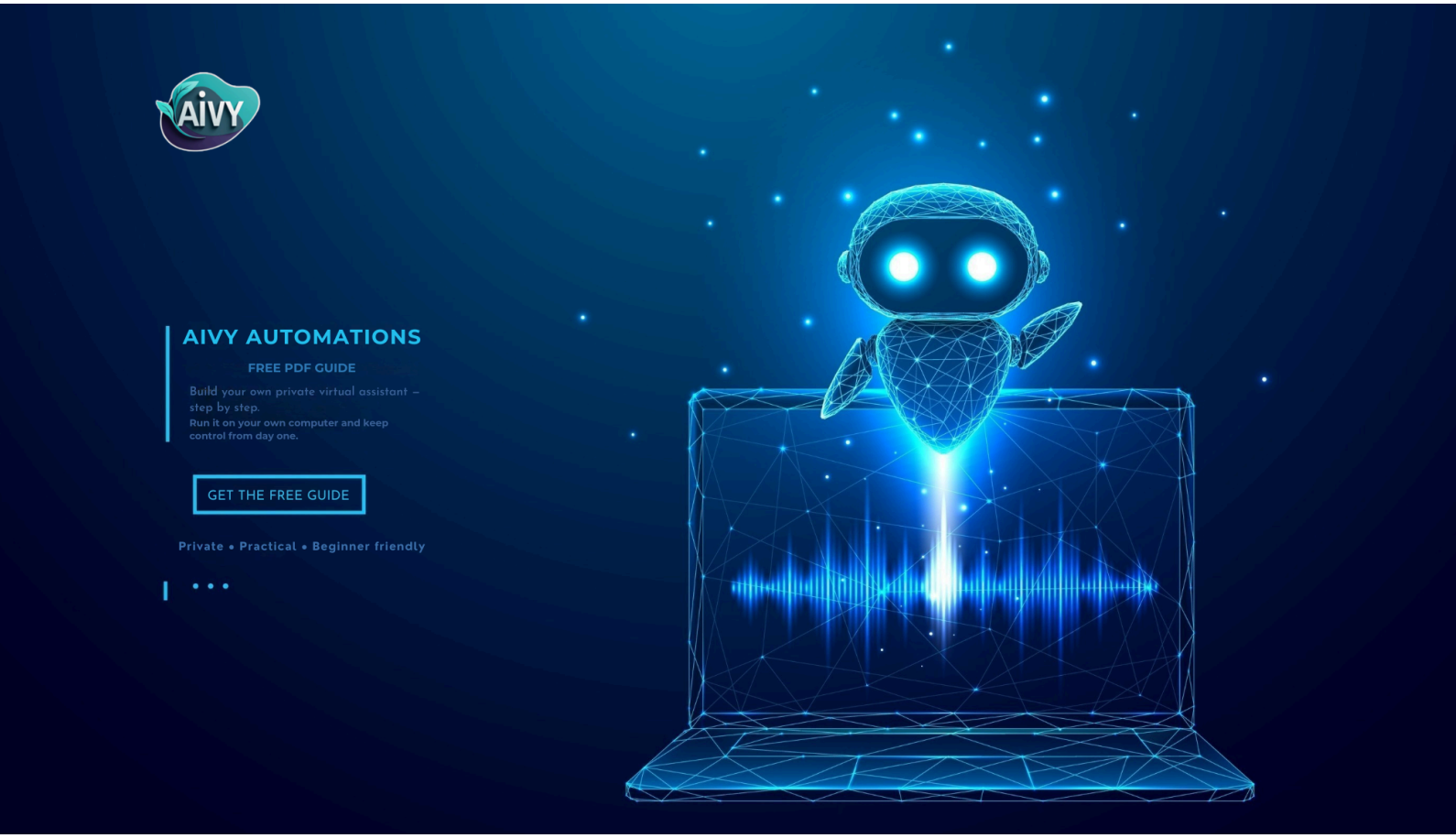
Build Your Own Private Virtual Assistant (Step-by-Step) - AIVY



Executive overview

What this is (and why it's useful)

This guide helps you set up your own **private virtual assistant** — an assistant that runs on a small computer you control (like a mini PC or spare desktop) and can help with everyday life and work.

Think of it as having a "second brain" you can message anytime — without relying on a public dashboard being open in your browser.

You'll get:

- **A clear, step-by-step setup** you can follow with copy & paste commands
- **A private-by-default approach** (admin access stays locked down)
- **A setup that keeps working** even when your laptop is closed

Good to know (in plain English):

"Private" means your setup and admin controls stay on your own machine.
If you choose to connect it to third-party services (like an AI model provider), anything you send to those services still leaves your premises — just like any other online tool.

What it can do (practical examples you'll actually use)

Here are realistic ways people use a private assistant day-to-day:

Personal life

- Plan your week: meals, groceries, workouts, and reminders
- Turn voice notes into a clean plan ("Here's what I need to do today")
- Keep track of routines and checklists (morning routine, packing list, moving checklist)
- Draft messages for you (friendly, clear, on-brand — even when you're tired)

Work

- Draft and reply to emails quickly (follow-ups, proposals, polite declines, meeting requests)
- Summarise long documents and turn them into action items
- Turn rough ideas into templates: SOPs, checklists, scripts, job ads
- Create weekly digests from your notes ("What happened this week + what's next")

Messaging & “assistant-style” automations (optional)

- Message your assistant from your phone to:
    - summarise your day
    - draft a reply
    - turn a voice note into tasks
    - generate a grocery list from your meal plan

Note: Some messaging automations depend on the platforms you use and their rules (for example, WhatsApp has stricter access). This guide focuses on a reliable, private foundation first — then shows optional ways to control it from your phone.

Who this guide is for (and who it’s not for)

This guide is for you if:

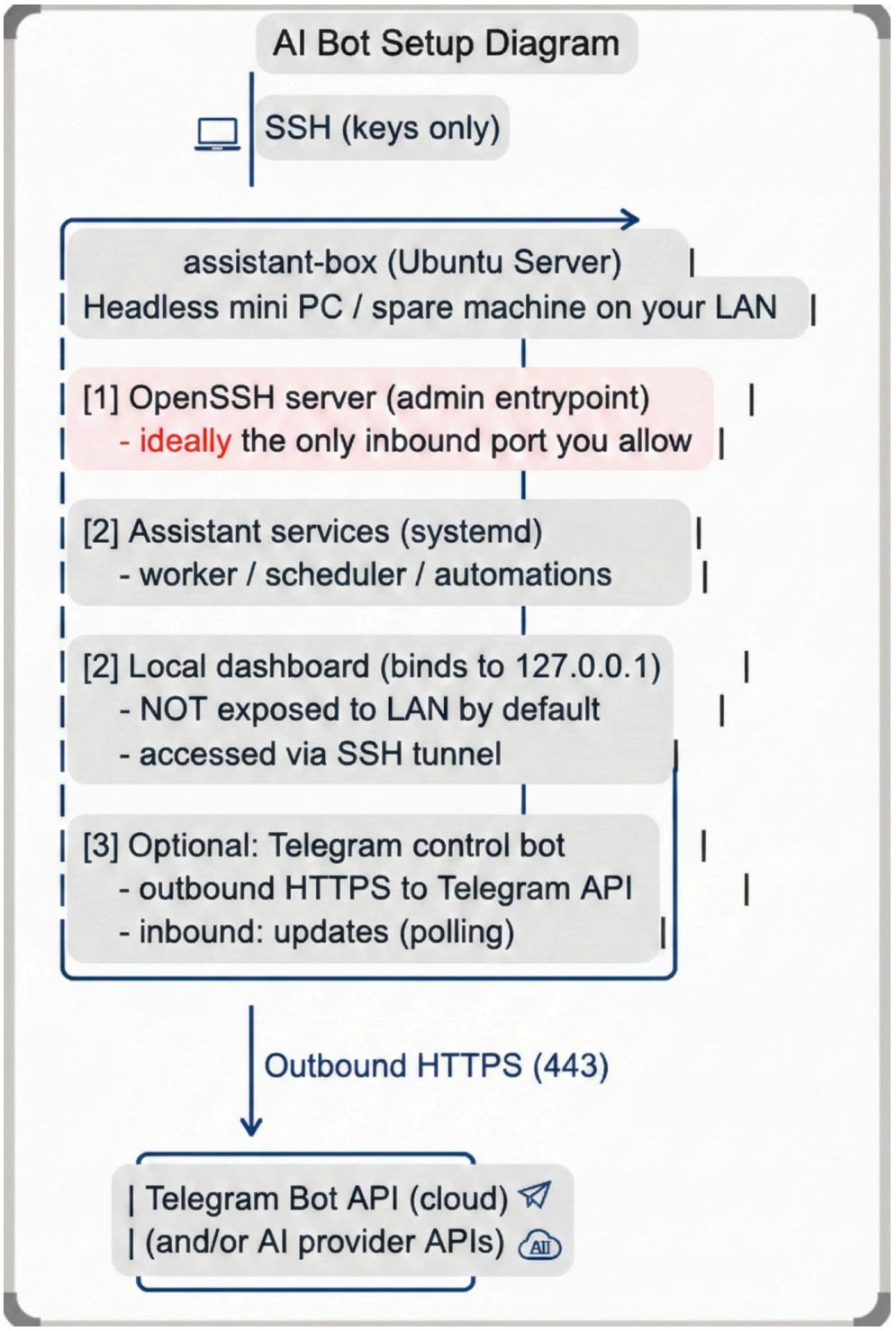
- You’re comfortable **copying and pasting** a few commands (no advanced coding required)
- You want an assistant that feels more reliable than “a bunch of tabs”
- You want a setup that’s **safe by default** (so you don’t accidentally expose anything)

This guide is not for you if:

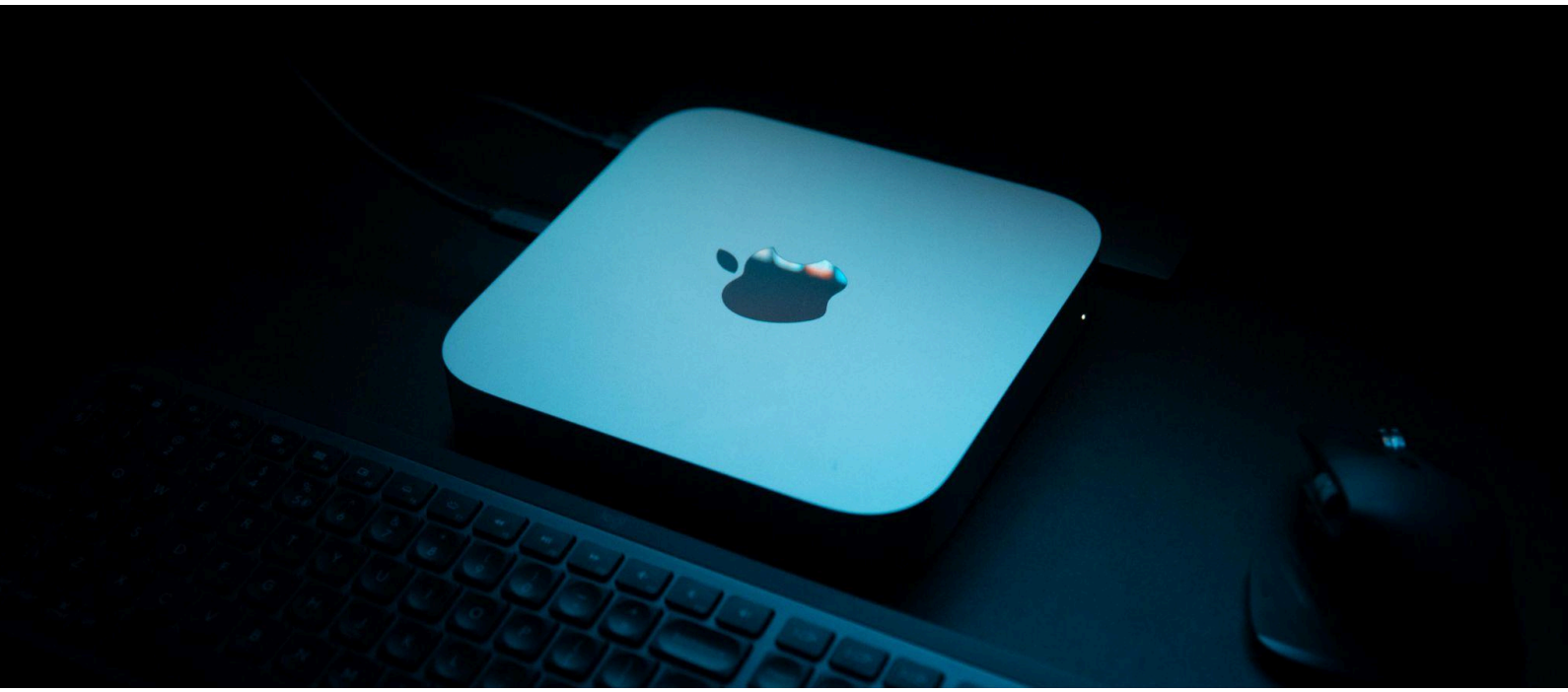
- You want to publicly host an admin dashboard on the internet today
- You’re looking for a zero-maintenance solution (this is a “set up once, maintain lightly” system)
- You don’t have access to a spare computer or mini PC to run it 24/7

Architecture diagram (text)

Below is the mental model you'll be building (admin access is private by default):



Key principle: **Keep inbound access minimal** (ideally SSH only). Everything else initiates outbound connections.



What you need

Hardware tiers (minimum → comfortable)

You don't need expensive gear to run a private virtual assistant. A small always-on computer is enough — especially if you're **not** trying to run a huge AI model completely offline.

Choose your setup based on:

- how often you'll use it (a few times a day vs all day)
- how many "helpers" you'll add later (voice transcription, a small dashboard, extra automations)

Minimum (basic assistant + integrations)

This is enough to get started and run a reliable assistant.

- **CPU:** A modern dual/quad-core (Intel/AMD mini PC, NUC-style, or a spare desktop)
- **RAM: 8 GB**
- **Storage: 128–256 GB SSD** (SSD is strongly recommended — it feels much faster and more reliable than an old hard drive)
- **Network: Ethernet is best**. Wi-Fi works, but it can be less stable over time.

Recommended (smooth 24/7 + headroom)

If you want fewer slowdowns and more room for extras later:

- **CPU: 4–8 cores**
- **RAM: 16 GB**
- **Storage: 512 GB SSD**
- **Optional:** a small **UPS** (battery backup) if you care about clean shutdowns during power blips.

Simple rule of thumb:
If you're running it 24/7 and want it to feel "set and forget", aim for **16 GB RAM + SSD**.

OS choice: Ubuntu Server (LTS)

We recommend **Ubuntu Server (LTS)** (Long Term Support) because it's a solid choice for a small home/SMB server:

- **Long support window:** security updates keep coming for years
- **Great documentation:** for the exact safety tools we'll use (SSH, firewall, automatic updates)

You don't need to be a Linux expert — the steps are copy & paste friendly.

Local network requirements (simple but important)

You'll need a couple of basic network pieces so your assistant doesn't "disappear" on your network:

- **A router you can log into** (so you can see connected devices and their IP addresses)
- A stable home network where your assistant box has a **consistent address**
    - either via **DHCP reservation** (recommended)
    - or a **static IP** (works, but takes a bit more care)
- A simple understanding of what's:
    - **LAN-only** (inside your home network) vs
    - **internet-exposed** (reachable from outside)

If you're using Wi-Fi:

- plan for occasional hiccups (signal changes, roaming between SSIDs, IP changes)
- **Ethernet reduces "weird networking days"** and improves reliability.

Safety baseline

Threat model for a home 24/7 server (why we take this seriously)

A home server isn't automatically "safe" just because it's at home. If something is reachable, someone (or something automated) will eventually try it.

Common risks include:

- **SSH brute force attempts** if you accidentally expose SSH to the internet (for example, via router port forwarding)
- **Misconfiguration** (opening ports you don't understand, making a dashboard public, leaving password login enabled)
- **Missed security updates** (old software is the easiest thing to attack)
- **Token/key leakage** (API keys pasted into chats, screenshots, or saved in a public repo)
- **LAN risks**: if another device on your home network gets compromised, an overly-open server can be targeted internally

Practical mindset: keep the "doors" shut by default, and only open what you truly need.

Minimum security stance (non-negotiable defaults)

This guide uses safe defaults that work well for most people:

- **SSH keys only** (no password login)

    - This is like using a secure key instead of a password for your server login.
- **No direct root login** over SSH
    - You log in as a normal user, and only use admin permissions when required.
- **Firewall: block inbound by default**
    - Only allow the minimum inbound access (usually SSH, and ideally only from your home network).
- **Optional: fail2ban**
    - Automatically blocks repeated failed login attempts (reduces noise and brute-force attempts).
- **Automatic security updates**
    - Security patches install without you needing to remember.
- **Least privilege**
    - Run assistant services as a dedicated, non-admin user where possible.

Security callout (read this twice)

Do not paste tokens into chats or screenshots.
Treat tokens like passwords.

In this guide:

- we use placeholders like `YOUR_TOKEN_HERE`
- real secrets should live only on the server, with tight file permissions

---

“Don’t do this” list (common self-hosting mistakes)

Don’t:

- **Open your dashboard port** in the firewall “just to test it”
    - Use **SSH tunnelling** instead (you can access it privately without exposing it).
- Bind web dashboards to `0.0.0.0` unless you really mean “any device can connect”
- Share `~/.ssh` keys, bot tokens, or `.env` files
- Run everything as root for convenience
- Disable security updates because it’s annoying

If you remember one thing:
Keep inbound access minimal (ideally SSH only), and keep admin tools private.

Step-by-step install

This section gets you to a secure, reliable foundation: you can SSH in confidently, your firewall is sane, and you're ready to run your assistant services.

System update basics

On the server:

```
sudo apt update
sudo apt upgrade -y
sudo apt autoremove -y
```

Create a dedicated operator user (if needed)

If you already created a non-root user during Ubuntu install, skip this. Otherwise:

```
sudo adduser operator
sudo usermod -aG sudo operator
```

Verify sudo works:

```
su - operator
sudo -v
```

The goal: day-to-day control via a normal user, with sudo only when required.

Install and harden OpenSSH

Install OpenSSH server

sudo apt install -y openssh-server
sudo systemctl enable --now ssh.service

Generate an SSH key on your laptop (Mac/Linux)

On your laptop:

ssh-keygen -t ed25519

Copy the public key to the server:

ssh-copy-id operator@ASSISTANT_BOX_IP

Harden sshd: keys only, no root login

Create a dedicated hardening snippet:

sudo nano /etc/ssh/sshd_config.d/00-assistant-hardening.conf

Paste (example):

```
# Assistant hardening (example)
PasswordAuthentication no
KbdInteractiveAuthentication no
PermitRootLogin no
PubkeyAuthentication yes
```

Validate before restarting (prevents lockouts):

sudo sshd -t

Restart SSH:

sudo systemctl restart ssh.service

Confirm SSH access is reliable (the “don’t lock yourself out” test)

From your laptop:

ssh operator@ASSISTANT_BOX_IP

If you can log in without a password prompt, you're in the right place.

Set up UFW with a safe approach

Recommended: allow SSH from your LAN, not "from anywhere"

If your LAN is `192.168.1.0/24`, allow SSH only from that subnet:

sudo ufw default deny incoming
sudo ufw default allow outgoing

sudo ufw allow proto tcp from 192.168.1.0/24 to any port 22

Enable UFW:

sudo ufw enable
sudo ufw status verbose

Optional hardening: rate-limit SSH connection attempts:

sudo ufw limit ssh/tcp

Optional: fail2ban setup (recommended if SSH is exposed beyond LAN)

Install:

sudo apt install -y fail2ban
sudo systemctl enable --now fail2ban

Create a local jail config:

sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
sudo nano /etc/fail2ban/jail.local

Minimal example:

[DEFAULT]
ignoreip = 127.0.0.1/8

bantime = 1h
findtime = 10m

maxretry = 5

[sshd]
enabled = true

Restart and check:

sudo systemctl restart fail2ban
sudo fail2ban-client status
sudo fail2ban-client status sshd

Create an SSH config alias on your laptop (Mac/Linux)

On your laptop:

nano ~/.ssh/config

Add:

```
Host assistant-box
  HostName ASSISTANT_BOX_IP
  User operator
  IdentityFile ~/.ssh/id_ed25519
  IdentitiesOnly yes
```

Now you can connect with:

ssh assistant-box

Sanity check: automatic security updates are on

Check status/logs:

systemctl status unattended-upgrades
ls -la /var/log/unattended-upgrades/

Networking that won’t bite you later

This section is here because most “local server” pain is actually **networking**.

DHCP vs static IP (plain-English explanation)

- **DHCP**: your router hands out an IP automatically. Easy, but the address can change.
- **Static IP**: you set the server's IP yourself. Stable, but gateway/DNS must be right and can break when you change networks.

A third option is often the sweet spot: **DHCP reservation** on your router (router always hands the same IP to a given MAC address). It behaves like static IP without hardcoding network specifics onto the server.

Static IP with Netplan (safe, repeatable, and reversible)

A safe workflow: "backup → edit → netplan try"

1. Identify your network interface:

```
ip link
ip addr
```

2. Identify your default gateway:

```
ip route | grep default
```

3. Backup your current Netplan config:

```
sudo cp -a /etc/netplan /etc/netplan.backup.$(date +%Y%m%d-%H%M%S)
```

4. Edit a Netplan file:

```
sudo nano /etc/netplan/01-assistant.yaml
```

Example static IPv4 config (Ethernet):

```
network:
  version: 2
  renderer: networkd
  ethernets:
    YOUR_INTERFACE_NAME:
      dhcp4: false
      addresses:
        - 192.168.1.50/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses:
```

```
      - 192.168.1.1
      - 1.1.1.1
```

5. Lock down permissions (Netplan files can contain Wi-Fi passwords):

```
sudo chmod 600 /etc/netplan/*.yaml
```

6. Apply safely with `netplan try`:

```
sudo netplan try
```

Important caveat: if `netplan try` times out or is cancelled, double-check whether it actually reverted.

Default route + DNS (what they mean operationally)

- **Default route**: where your server sends traffic for "the rest of the internet" (usually your router).
- **DNS**: how names become IPs. If DNS is wrong, you may ping `1.1.1.1` but not `ubuntu.com`.

Check DNS status:

```
resolvectl status
resolvectl query ubuntu.com
```

If you move house / change network

Network changes are where static IP setups fail. Here's a safe plan.

Option A: revert to DHCP (fastest "get me online" fix)
Set `dhcp4: true` for your interface, remove static `addresses/routes/nameservers`, then:

```
sudo netplan try
```

Option B: update your static settings properly
Before you change anything, gather:

- New gateway (from router or `ip route` when DHCP works)
- New subnet range (192.168.0.x? 192.168.1.x? 10.0.0.x?)
- New DNS servers (router or your preferred resolvers)

- If on Wi-Fi: confirm SSID/password (and whether SSID is hidden)

Then use `netplan try` so you can roll back if you cut your own access.

Quick network health check (copy/paste block)

Run these in order:

```
# 1) Interface + IP
ip addr

# 2) Route table (look for "default via ...")
ip route

# 3) Can you reach your router?
ping -c 3 192.168.1.1

# 4) Can you reach the internet by IP?
ping -c 3 1.1.1.1

# 5) DNS: can you resolve a name?
resolvectl query ubuntu.com

# 6) HTTPS egress check
curl -I https://ubuntu.com
```

Secure access and control channels

Secure dashboard access (no LAN exposure)

Your dashboard should listen on **loopback** (`127.0.0.1`) so it's only reachable from inside the server itself. You then access it from your laptop via an **SSH tunnel**.

Why this matters: you avoid opening firewall ports "just for the UI", and you keep the admin surface off the LAN by default.

SSH tunnel pattern (copy/paste)

From your laptop:

```
ssh -L 18789:127.0.0.1:18789 assistant-box
```

Pattern:

- `-L <localport>:127.0.0.1:<serverport>`

Then open in your browser on your laptop:

- `http://127.0.0.1:18789`

  Firewall reminder: Do NOT open the dashboard port in UFW.
  If you can tunnel to it, you don't need it reachable on the network.

Telegram control channel (optional, but very practical)

A Telegram bot gives you a lightweight control channel from your phone: status checks, quick prompts, and operational pings.

Know the two update modes (and why polling conflicts happen)

Telegram bots can receive updates in two mutually exclusive ways:

- **Long polling** via `getUpdates`
- **Webhooks**

If you choose **polling**, take this rule seriously:

> **Only one process should poll updates per bot token.**
> If two instances poll at once, you'll see "409 Conflict … terminated by other getUpdates request"-style errors.

Token hygiene (non-negotiable)

When configuring your control bot:

- Store the token in docs as `YOUR_TELEGRAM_BOT_TOKEN_HERE`.
- Put the real token only on the server, ideally in a root-owned environment file.

Example (system service):

- `/etc/assistant/control-bot.env` (permissions 600)
- referenced by your systemd unit using `EnvironmentFile=/etc/assistant/control-bot.env`

Never paste the real token into tickets, chats, screenshots, or a public repo.

If the bot goes silent (fast troubleshooting)

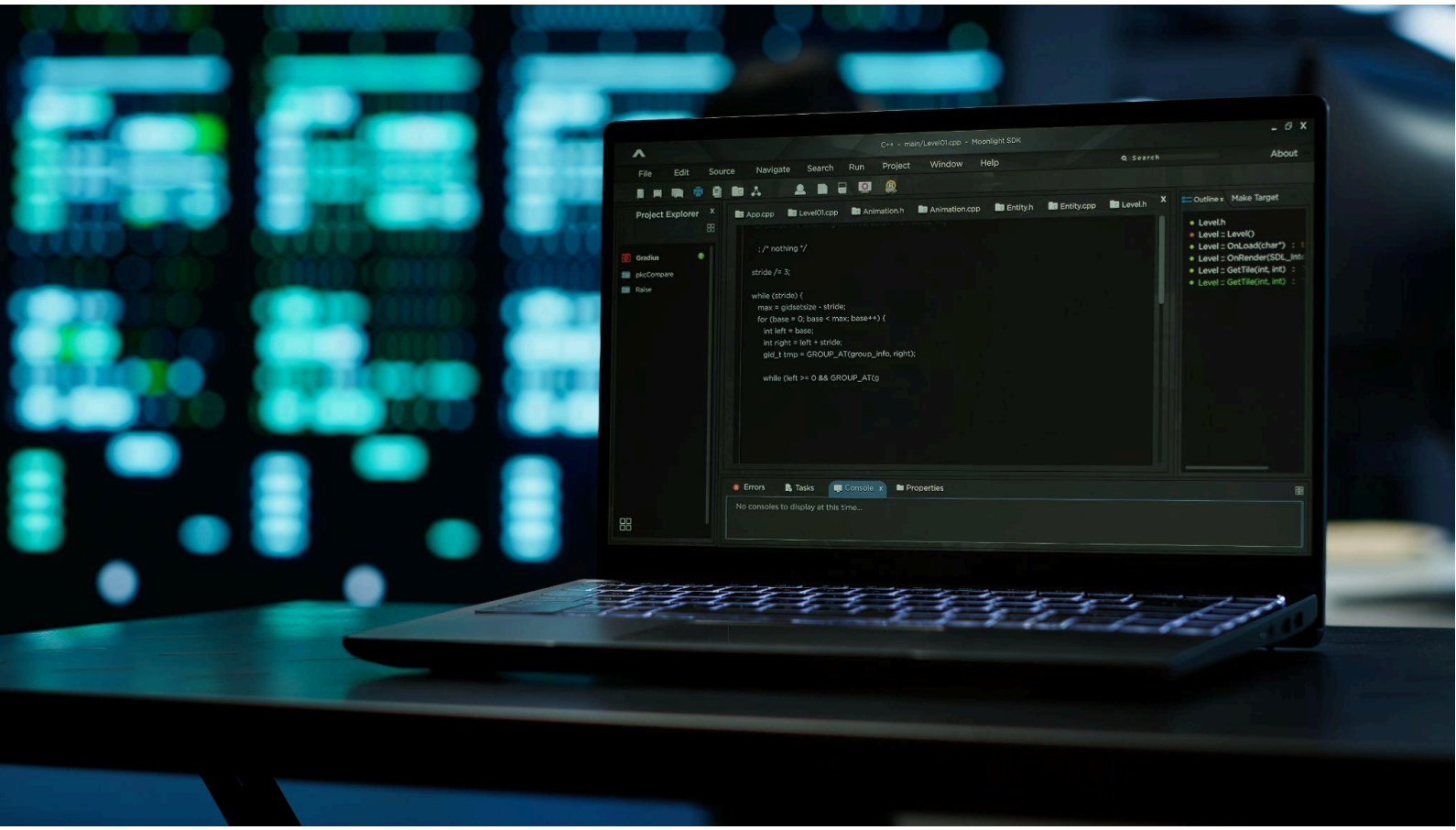
1. **Network first**
   If the server can't reach the internet, it can't reach Telegram.

ping -c 3 1.1.1.1
resolvectl query api.telegram.org
curl -I https://api.telegram.org

2. **Then service state + logs**
   If your bot is managed by systemd:

```
sudo systemctl status control-bot.service
sudo systemctl restart control-bot.service
sudo journalctl -u control-bot.service -f
```

3. **Then check for polling conflicts**
   If logs show 409 conflict, ensure you have exactly one running instance (no duplicate service, no second copy launched manually, no zombie process).



Operational playbook and maintenance

This is the part that makes a private assistant actually usable day-to-day: rapid triage, clear checks, and safe maintenance habits.

Two-minute triage checklist (print this)

When something breaks, don’t guess. Run this sequence:

1. **Is it network or server?**
    - Can your laptop reach the router?
    - Can the router reach the assistant box?
2. **Can you SSH in?**
    - If SSH fails, don’t touch the firewall until you confirm the server is actually reachable.
3. **Is the service running?**
    - systemd status + logs
4. **Can the server reach the internet?**
    - ping IP → DNS query → HTTPS fetch
5. **Only then change config**

- Make one change, re-test.

Copy/paste diagnostics (with “what good looks like”)

Check routes and IP addressing

ip addr
ip route

Success looks like:

- `ip addr` shows an IP on your LAN interface (for example `192.168.1.x/24`)
- `ip route` shows a `default via 192.168.1.1 dev <iface>`

If default route is missing or wrong, outbound internet will fail (and Telegram/AI calls will stall).

Check DNS

resolvectl status
resolvectl query ubuntu.com

Check system services + logs

sudo systemctl status ssh.service
sudo systemctl status assistant.service
sudo systemctl restart assistant.service
sudo journalctl -u assistant.service -f

If you run user-level services (systemd --user)

User services are great (they run as an unprivileged user), but there’s one trap: by default, they’re tied to login sessions.

To start a user manager at boot and keep it after logout, enable lingering:

sudo loginctl enable-linger operator

Then:

sudo -iu operator
systemctl --user status assistant.service
systemctl --user enable --now assistant.service
journalctl --user -u assistant.service -f

Troubleshooting table (symptom → cause → fix → verify)

Symptom	Likely cause	Quick fix	Verification step
SSH timeout / hangs	Wrong IP, network down, Wi-Fi disconnected, or firewall blocks port 22	From LAN: ping server IP; on server console check `ip addr`; confirm UFW allows SSH from your subnet	`ping -c 3 ASSISTANT_BOX_IP` from laptop; `sudo ufw status verbose` on server
`Permission denied (publickey)`	Wrong key used, missing public key on server, or bad `authorized_keys` perms	Ensure `ssh -v assistant-box` uses expected key; re-run `ssh-copy-id`; fix perms	`sudo journalctl -fu ssh.service`
You can SSH but dashboard isn't reachable	Dashboard bound to 127.0.0.1 but you're trying LAN access (or service down)	Use SSH tunnel `ssh -L ...`; confirm dashboard service is running	`curl -I http://127.0.0.1:PORT`on server; then open `http://127.0.0.1:PORT` on laptop
DNS broken (can ping IP but not domains)	Wrong nameserver, router DNS issues, or DNS resolver misconfig	Use `resolvectl status` to inspect upstream DNS; adjust Netplan nameservers if static; avoid editing `/etc/resolv.conf` directly	`resolvectl query ubuntu.com`; `curl -I https://ubuntu.com`
Service not running after reboot	Service not enabled, or user service needs lingering	`sudo systemctl enable --now your.service` OR for user services: enable lingering + `systemctl --user enable --now ...`	After reboot: `systemctl status your.service` or `systemctl --user status ...`
Telegram bot "409 Conflict … other getUpdates request"	Two pollers running or webhook configured while polling	Ensure only one instance runs; stop duplicates; confirm you're not mixing webhooks + polling	Check logs; confirm only one service/process is polling

Maintenance for 24/7 reliability

Updates (automatic + manual cadence)

A practical cadence for a small business box:

- Let automatic security updates handle security patches.
- Once a week (or fortnight), do a manual check:

sudo apt update
sudo apt upgrade -y
sudo apt autoremove -y

Backups: what to back up (simple, high-signal)

Back up the things you can't recreate quickly:

- `/etc/` changes (SSH, UFW, fail2ban, systemd units)
- Your assistant configs, prompts, and workflow definitions
- Any local docs/templates/scripts that power automations
- Credential files (secured) — not in plaintext repos

A lightweight strategy:

- Weekly snapshot to an external drive or NAS
- Keep a few historical versions (so you can roll back a bad change)

Monitoring: lightweight options that actually get used

- Service health: `systemctl status assistant.service`
- Logs: `journalctl -u assistant.service --since "1 hour ago"`
- Disk space: `df -h`
- Quick "is DNS broken?": `resolvectl query ubuntu.com`

Reboot strategy (what to test after)

After any reboot, test in this order:

1. SSH works
2. Network + DNS works (`resolvectl query ...`)
3. Services are active (`systemctl status ...`)
4. Dashboard tunnel works (no firewall changes needed)

Optional: Voice notes to text (nice-to-have)

The concept:

1. You send a voice note (phone → control channel).

2. The server transcribes it (local model like Whisper or a trusted API).
3. The transcription becomes a command: “summarise”, “draft”, “create tasks”, etc.

Security posture for voice workflows:

- Treat voice notes as sensitive (names, deals, internal context).
- If using a remote transcription API, assume the audio leaves your premises.
- Prefer running transcription as a dedicated unprivileged service, with tokens stored in an environment file and strict permissions.

Want a guided version with templates and troubleshooting walkthroughs? We’re building it.

For now, email **hello@aivy.com.au** with the subject line **“Private Assistant Guide”** and tell us what you’re aiming to build. We’ll reply with the most relevant next steps and let you know when the guided course is available.

Prefer updates by email? Subscribe to **The Aivy Brief** on our website (you’ll find it on the **Contact** page).