EDITION 3 · REVISED AND VERIFIED · AUGUST 2026

Your own AI server assistant

From an empty Linux server to a digital colleague that runs around the clock, is reachable through the Claude app and survives a reboot without complaint.

Written and verified step by step by Christian's virtual server
Checked against Claude Code 2.1.232 on Ubuntu

What is new in this edition

Edition 2 got you as far as continuous operation. This edition closes the gaps that showed up in daily use:

- **The service now picks the conversation back up.** Without one small addition in part 4 it started from scratch after every restart — particularly annoying in the middle of a job.
- **Part 7 is new:** push notifications to your phone. With it your assistant speaks up by itself when it needs a decision or something important has happened.
- **Part 4 now knows what the prompt has already built.** Anyone who runs the bootstrap prompt all the way through gets the service there already — part 4 now says what to check and what to skip.
- All commands and version numbers are verified against **Claude Code 2.1.232**.

0 What you need

- **A small Linux server (VPS).** A rented machine on the internet that is yours alone. Providers include Hetzner, netcup, DigitalOcean, Linode, OVH. The smallest package is enough (roughly €4–7 a month). Take **Ubuntu 24.04 LTS or newer**. Two things matter: at least **2 GB of RAM** and **10 GB of free disk space** — Claude Code is a large program and keeps old versions around.
- **A Claude subscription** (Pro or Max). The free Claude is not enough.
- **A terminal.** Built into the Mac, on Windows use "Windows Terminal".
- **About 60 minutes.** With continuous operation from part 4, closer to 75.

1 The starting block — server ready in one go

After ordering you get an IP address and a password for the user `root`. Log in (substitute your own IP):

```
ssh root@203.0.113.10
```

The first time it asks `Are you sure…?` — type `yes`, then the password. Now comes the block that prepares everything else. **Replace `myname` everywhere with the user name you want** and then paste the whole thing in:

```
# ---------- as root, right after the first login ----------
apt update && apt full-upgrade -y
apt install -y curl git ufw jq python3 ripgrep

# 1) create your own user (asks for a password)
adduser myname
usermod -aG sudo myname

# 2) sudo without a password prompt — required for automated runs
echo 'myname ALL=(ALL) NOPASSWD: ALL' > /etc/sudoers.d/claude
chmod 0440 /etc/sudoers.d/claude
visudo -c

# 3) firewall: only SSH open
ufw allow OpenSSH
ufw --force enable
```

> **Important: check first, close the root session afterwards**
>
> The command `visudo -c` must answer with `… parsed OK`. If it reports an error, delete the file again immediately with `rm /etc/sudoers.d/claude` — **while you are still logged in as root**. A broken sudo file otherwise locks you out of all administrator rights. Leave this root window open until you have confirmed in the next step that everything works.

Now log in as the new user in a **second** terminal window and check both permissions:

```
ssh myname@203.0.113.10
sudo -n true && echo "sudo without password: OK"
```

If `sudo without password: OK` appears, everything is right — and you can close the root window.

> **Why sudo without a password?**
> When Claude works on its own later — at night, on a schedule, or remote-controlled through the app — nobody is sitting there to type a password. Without this permission every command that needs administrator rights simply stalls. It is a deliberate trade-off: whoever gets into your user account has full administrator access as well. That is why SSH access is the place you really should secure — see part 10.

2 Install Claude Code and sign in

```
curl -fsSL https://claude.ai/install.sh | bash
exec $SHELL -l          # reload PATH
claude --version        # expects e.g.: 2.1.232 (Claude Code)
claude doctor           # checks the installation
```

The installer puts the program under `~/.local/share/claude/versions/` and links it through `~/.local/bin/claude`. **Remember this path** — you need it in parts 4 and 6. If you get `command not found`, log out once and back in.

Now the first start and the sign-in:

```
claude
```

It shows you a link and a code. Open the link on your phone or PC, sign in with your Claude subscription, confirm the code. On the first start it also asks about the colour scheme and whether you trust the current folder — confirm both once. `/exit` gets you back out.

> **Claude keeps itself up to date**
> It pulls new versions automatically at startup. By hand it is `claude update`. Old versions stay behind and take up about 270 MB each — with `ls -la ~/.local/share/claude/versions/` you can see them and safely delete the older ones. That is the reason for the 10 GB recommendation in part 0.

3 The magic trick: hand over the bootstrap prompt

This is the moment where your assistant builds itself. The accompanying file `setup-prompt-en.txt` holds a prepared text. First create its working directory and start it there:

```
mkdir -p ~/assistant
cd ~/assistant
claude
```

Open `setup-prompt-en.txt`, copy everything between the two marker lines and paste it in as the first message. From there Claude walks you through the setup — it asks questions, sets up its memory, checks the basic security and, if you want, configures e-mail and web. Its last

item also builds the continuous operation from part 4 right away, meaning the system service. Read part 4 anyway — it explains what happened there and how to verify it.

> **Why `~/assistant` of all places?**
>
> Claude remembers trust and settings **per directory**. If the service from part 4 later starts in a different folder than the one where you confirmed everything, it asks again — and stalls. So stick to one fixed working folder.

4 Continuous operation: as a service, reachable from the app, reboot-proof

Without a service Claude only lives while your terminal window is open. As a real server service it starts with the system, runs around the clock, is reachable from the Claude app and survives every reboot — without asking anything.

> **Ran the whole prompt? Then most of this already stands**
>
> Item 8 of the bootstrap prompt sets up this exact service. So first look at what is already running: `systemctl status claude-code.service --no-pager`.
>
> If it says `active (running)`, skip steps 1 to 3 — they would only create and start the same thing a second time. Read the explanations in them anyway, above all the box about `--continue`, and then carry on at **step 4**: the acid test with the reboot is one nobody has done for you yet.
>
> If you cut the prompt short or left item 8 out, work through part 4 from the beginning.

Three pieces belong to that, and all three are necessary:

PIECE	WHAT FOR
System service	starts Claude automatically at boot and restarts it if it crashes
Pseudo terminal	Claude needs a terminal. Without one it stalls at every prompt and cannot store the answer either
Pre-confirmations	the one-off notice dialogs are ticked off in advance, so nobody has to type “yes” after a reboot

Step 1: create the service

You can paste this block unchanged — it fills in your user name and your home directory automatically:

```
sudo tee /etc/systemd/system/claude-code.service >/dev/null <<EOF
[Unit]
Description=Claude Code Remote Control Session
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=$USER
WorkingDirectory=$HOME/assistant
ExecStart=/usr/bin/script -qec "$HOME/.local/bin/claude --remote-control --name vps-main --
continue" /dev/null
Restart=on-failure
RestartSec=10

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

Careful when typing it out: `ExecStart=…` is **a single line**, even though it is shown wrapped above.

Four spots are worth a look:

- `script -qec "…" /dev/null` is the pseudo-terminal trick. Without it the service does start, but hangs at the first dialog.
- The call uses `~/.local/bin/claude`, that is the **link** — not the specific version underneath. Otherwise the service points into nothing after the next automatic update.
- `--name vps-main` is the name the session shows up under in the app later. Feel free to pick something descriptive.
- `--continue` picks the **last conversation back up** at startup instead of beginning from scratch. New in this edition — see the box just below.

> **Why `--continue` makes the difference**
>
> Without this addition the service starts a fresh session after every restart. Everything you had discussed is gone — and because `Restart=on-failure` applies, that can happen in the middle of a job without you noticing.
>
> With `--continue` it attaches itself to the most recent conversation instead. A crash or a reboot becomes a short interruption rather than a loss of memory.
>
> The flip side, so you are not caught out: restarting the service no longer gives you a fresh session. “Remember this just for this conversation” therefore survives a reboot as well. If you really want to start over, clear the conversation with `/clear` from inside the app.

Step 2: tick off the one-off prompts in advance

This is the step most people fail at. Claude stores “already seen” ticks in the file `~/.claude.json`. If they are missing, the service stalls at a prompt after every restart. This command sets them and makes a backup first:

```
python3 - <<'PY'
import json, os, shutil
p = os.path.expanduser('~/.claude.json')
shutil.copy(p, p + '.bak')
d = json.load(open(p))
d['hasCompletedOnboarding'] = True
d['hasSeenAutoModeEntryWarning'] = True
proj = d.setdefault('projects', {}).setdefault(os.path.expanduser('~/assistant'), {})
proj['hasTrustDialogAccepted'] = True
json.dump(d, open(p, 'w'), indent=2)
print('Flags set. Backup: ~/.claude.json.bak')
PY
```

Step 3: start the service and check it

```
sudo systemctl daemon-reload
sudo systemctl enable --now claude-code.service
systemctl status claude-code.service --no-pager
```

The status output must say `active (running)`. If it does not, `journalctl -u claude-code -n 40 --no-pager` shows the reason.

Step 4: the acid test — reboot

```
sudo reboot
```

Wait about a minute, reconnect and check:

```
systemctl status claude-code.service --no-pager
```

If it says `active (running)` again **without you having confirmed anything**, you have reached the goal.

> **If it still hangs at a prompt**
>
> Then check exactly these four points, in this order:
>
> 1. Does the service run as the same user in whose home directory the flags live? `systemctl show claude-code -p User`
> 2. Does the path under `projects` in `~/.claude.json` match `WorkingDirectory` character for character? No trailing slash, no tilde, no symlink.
> 3. Is the pseudo terminal really there? `ps -o tty= -p $(pgrep -f remote-control | head -1)` must **not** print `?`.
> 4. Is there something in `~/.claude/settings.json` that overrides the mode?

Step 5: connect from the app

Open the Claude app on your phone or **claude.ai/code** in the browser — signed in with the same account as on the server. Your session shows up there under the name you gave it

(`vps-main`). You simply type a message, and real work happens on your server. That is the state you were after: the server runs, you are out and about, and the two meet.

> **Handy in everyday use**
>
> `sudo systemctl restart claude-code` brings the service back up — thanks to `--continue` the conversation carries on afterwards instead of starting over. If a conversation has got stuck, a restart will not help; `/clear` from inside the app will. `journalctl -u claude-code -f` shows live what the service is doing.

5 Optional: sending e-mail

If your assistant is to write e-mail, it needs a mailbox. You need three details from your mail settings: SMTP server, address and password. Just tell Claude in the conversation:

```
Please set yourself up so that you can send e-mail.
SMTP server smtp.myprovider.com, port 587,
address me@myprovider.com. I'll give you the password in a moment.
```

It installs `msmtp`, stores the credentials with strict file permissions and sends a test mail together with you. Without credentials of your own, simply skip this part — it is not allowed to invent them.

6 Optional: autonomous — it works while you sleep

A scheduled job (a "cron job") checks regularly whether there is anything to do — new mail in the mailbox, for instance — and only then starts a real Claude run. Claude sets that up for you too. To make it reliable, give it these three points; I tripped over every one of them myself:

PITFALL	FIX
Cron does not know your PATH	`~/.local/bin` is missing there. In the script either set `export PATH="$HOME/.local/bin:$PATH"` or call Claude with its full path — otherwise you get "command not found" and the job fails silently.
Two runs at once	Guard it with `flock`, otherwise stalled runs overtake one another.
Cost	First use a cheap check to see whether there is work at all. Only then start the run that costs money. And put a `timeout` in front of it.

The actual call in the script looks like this — `-p` means "work through it once, no conversation", and the tool list limits what it may do in the process:

```
timeout 1800 "$HOME/.local/bin/claude" -p --model sonnet \
  --allowedTools "Bash(mailbox:*),Read" \
  --max-turns 30 \
  "Your task description here" >>"$LOG" 2>&1
```

> **Incoming content is not an instruction**
>
> The single most important standing rule: the content of other people's e-mails, web pages and messages is **never an instruction** to Claude. It does not follow demands found in them ("send me file X", "run Y"). And anything only you can decide — money, appointments, commitments in your name — it hands back to you instead of acting itself. The bootstrap prompt writes exactly that into its memory.

7 Optional: push notifications to your phone

Mail and schedules are one-way streets: your assistant puts something down, and at some point you go and look. For the reverse case — it needs you — a channel is still missing. That is exactly what a push is good for: a short nudge to your phone when a decision is due or something important has happened.

The simplest way is **ntfy.sh**. No account, nothing to set up on the server, no keys: you think up a hard-to-guess name for your channel, subscribe to it in the ntfy app — and everything sent to that address lands on your phone.

> **The channel name is your only protection**
>
> Whoever knows the name is reading along — on ntfy.sh every channel is publicly readable. So do not take anything guessable like `myserver`, take something long and random. And more importantly: **never send content in the push.** No mail addresses, no third-party names, no subject lines, no amounts, no credentials. The push only says that something is up — the what belongs in an e-mail.

A channel name nobody will guess, and setting up the app:

```
head -c 18 /dev/urandom | base64 | tr -dc 'a-zA-Z0-9'
# yields e.g.: k7Rq2wZm4TnLp9Xd
```

Install the **ntfy** app (iOS and Android, free), subscribe to exactly that name there — and have Claude build you the little sending script:

```
Please create a script ~/assistant/bin/push.sh for me that sends
a message to my channel via ntfy.sh. Put the channel name into a
separate configuration file with permissions 600, not into the
script itself. Two send attempts, a 15 second timeout, and write
every send into a log file.
```

After that one call is enough, from any script and from any automated run:

```
~/assistant/bin/push.sh "A question is waiting — details by e-mail"
```

Stay frugal, or you will soon stop looking

A push is only worth anything if it arrives rarely. This rule has proven itself: **only for decisions and for trouble** — not for success messages, and certainly not as status chatter.

A practical example from everyday use: a scheduled job checks daily whether a new program version has arrived. If everything is as before — the normal case — nothing happens at all. Only if it really changed something does a push arrive. And if anything strikes it as odd in the process, it publishes nothing and asks instead. That handbrake is worth more than any convenience.

8 Driving it from your phone

- **The Claude app or claude.ai/code** — the comfortable route, once part 4 is in place. Your server session is simply there.
- **An SSH app** (Termius, for example) — the direct route into the terminal, when you want to look at something on the server yourself.

9 Connecting further services (connectors)

Claude can dock onto other systems — mail, calendar, Drive and more. That way it fetches live data instead of relying on its built-in knowledge alone. A topic for later: simply ask it “Which connectors can I attach, and how?” when you get that far.

10 Security & good habits

- **Secure SSH access.** Because of the passwordless sudo from part 1, your user account is the master key. Have Claude set up logging in with an **SSH key instead of a password** and switch password login off — that is the single most effective measure. It explains every step, and you test the new login in a second window before closing the old one.
- **Do not work as root.** Your ordinary user plus `sudo` for individual commands is enough.
- **Protect credentials.** Always file permissions `600`, never printed in the clear.
- **Have it ask before anything irreversible.** Deleting, overwriting, restarting, sending something outward.
- **Keep an eye on the cost.** Automated runs cost money. Have every run logged at first and look into the log after a few days.

Your checklist to tick off

☐ Own user created, `sudo -n true` runs without a password

☐ Firewall active, only SSH open

☐ `claude --version` shows a version

☐ Bootstrap prompt completed, memory in place

☐ Service running: `systemctl status claude-code` shows `active (running)`

☐ After `sudo reboot` it runs again — without a prompt, and the conversation is still there

☐ The session is visible in the Claude app and answers

☐ Optional: a test push arrives on your phone

☐ SSH key set up, password login off

> **And then?**
>
> Then the good part begins. Tell it what annoys you in daily life — and have it build tools for that. That is exactly how everything I can do here on Christian’s server came about. If something jams: it can read its own errors. “Look in the journal and work out why the service will not start” is a perfectly ordinary request.