Self-Host Openclaw

Self-Host Openclaw (Clawdbot) in a Proxmox VM (The Secure Way)

Self-Host Openclaw (Clawdbot) in a Proxmox VM (The Secure Way)

Self?Hosting Openclaw (Clawdbot) on Proxmox VM – Secure Setup

This guide walks through a security-focused Openclaw (formerly Clawdbot/Moltbot) deployment on Proxmox using a dedicated Ubuntu VM, a non-privileged user, and a localhost-only gateway.

---

1. Why a VM and not LXC

Openclaw is an AI agent with shell access, persistent storage, and plugins, so a compromise of the agent can become a compromise of the host if isolation is weak.

- LXC containers share the host kernel; a container escape could give an attacker access to the Proxmox host.
- A Proxmox VM uses hypervisor isolation: if the VM is compromised, you can destroy it and your Proxmox host remains protected.

Additional prerequisites (strongly recommended):

- Up-to-date Proxmox host with QEMU agent support.
- Router firewall + IDS/IPS as appropriate.
- Regular backups of important data (treat this VM as disposable).

---

2. Create the Proxmox VM

1. In Proxmox, create a new VM, name it e.g. `openclaw`.
2. OS:
   - Use Ubuntu Server 24.04 ISO (or similar) as the installation media.
3. System settings:
   - Machine: default (q35) is fine.
   - BIOS: default.
   - CPU:

    - Cores: at least 2 vCPUs.
    - Type: `host` to expose all CPU features.
  - Memory:
    - Give more than the minimum (e.g. 4–8 GB depending on what you'll run).
    - Disable ballooning; Node.js + plugins can use more RAM unexpectedly, and ballooning can cause instability.
  - Disk: 32 GB+ recommended (depends on logs, plugins, etc.).
4. Enable the QEMU guest agent for better integration.
5. Finish the wizard and start the VM.

Inside the Ubuntu installer:

- Accept most defaults.
- On the "SSH" step, **enable OpenSSH server** so you can use your own terminal later instead of the web console.

---

3. Initial Ubuntu hardening and updates

From your workstation:

```
ssh youruser@<vm-ip>
```

Then:

```
sudo apt update && sudo apt upgrade -y
```

Apply security updates before installing anything else.

---

4. Create a dedicated Openclaw user

Never run Openclaw as `root`. Use a separate, unprivileged user as a sandbox.

```
sudo adduser openclaw
```

Follow prompts for password; optional extra info can be blank

Optionally restrict `sudo` for this user (recommended: no sudo at all). Then switch:

```
sudo su - openclaw
```

All further Openclaw-related steps run as this **openclaw** user unless otherwise noted.

---

5. Add swap (for Node.js compilation)

Openclaw is a Node.js package and can compile plugins and dependencies during install; lack of RAM/swap can cause failures.

As `root`:

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

Verify:

```
swapon --show
```

---

6. Install Node.js and Openclaw

Still on the VM:

1. Install Node.js (example using distro or NodeSource; adapt to your preferred method). The video uses a repository add + install flow.

Example (NodeSource, adjust version as needed):

```
curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -
sudo apt install -y nodejs
```

2. Switch to the dedicated user and install Openclaw globally:

```
sudo su - openclaw
npm install -g openclaw
```

(Use the actual package name used in the docs; the video installs the “cloudbot/Openclaw” CLI globally via npm.)

---

7. Telegram bot setup

The example integration in the video uses Telegram, but you can map this process to any supported chat client.

From your phone/desktop Telegram client:

1. Talk to `@BotFather`.
2. `/newbot` → choose a name and username.
3. Copy the **HTTP API token**; you'll paste it in the Openclaw onboarding later.
4. In BotFather:
   - Disable group privacy so the bot can read all messages in groups (if you want group usage).
5. Get your personal Telegram user ID:
   - Talk to `@userinfobot` (or similar), send `/start`, copy the numeric user ID.

Keep:

- Bot API token.
- Your Telegram user ID.

---

8. Run the Openclaw onboarding wizard

From the `openclaw` user shell:

```
openclaw init
```

or the appropriate CLI command that starts the onboarding wizard

During the wizard (as shown in the video):

1. Choose **manual configuration**.
2. Select your LLM provider:
   - Example in the video: Anthropic, model `claude-3-opus-4.5` (adjust for current names/pricing).
   - Paste your API key.
3. **Gateway binding (critical security step)**:
   - Set the gateway bind address to **loopback only** (e.g. `127.0.0.1:PORT`).
   - This ensures the HTTP gateway listens only on the VM itself, with **no open ports to your LAN or the internet**.
4. Enable Telegram integration:
   - Paste the Telegram bot token.
   - Enter your Telegram user ID if requested as an initial allowed user.
5. Skip skills/hooks initially to keep the surface area small.

The wizard may attempt to install a systemd service and report a message like "system user services unavailable" at the end. We'll fix that manually next.

9. Fixing the `USER_ID` environment gotcha

In the video, the service failed to start until an environment variable for the current user ID was exported.

As the `openclaw` user:

```
echo "export USER_ID=$(id -u)" >> ~/.bashrc source ~/.bashrc
```

If Openclaw expects a different variable name (e.g. `CLOUD_USER_ID` or similar), match what the docs/onboarding output indicates; the video exports the current user ID so the Node environment/compiler knows which user it is running as.

---

10. Create a systemd service manually

Since the onboarding script's service creation is unreliable, create a systemd unit yourself so Openclaw auto-starts on boot and survives logout.

As `root`:

```
sudo nano /etc/systemd/system/openclaw.service
```

Example unit file (adapt command/paths to your setup):

```
[Unit]
Description=Openclaw AI Agent
After=network.target

[Service]
Type=simple
User=openclaw
WorkingDirectory=/home/openclaw
Environment=USER_ID=%i # or the variable you need, or drop if not required
ExecStart=/usr/bin/npx openclaw start
Restart=on-failure
RestartSec=5

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

Save and reload systemd:

```
sudo systemctl daemon-reload
sudo systemctl enable --now openclaw.service
sudo systemctl status openclaw.service
```

Confirm the service is **active (running)**.

---

11. Run the built?in security audit

Openclaw ships with a security audit command that checks configuration issues.

As `openclaw`:

```
openclaw audit
```

or the appropriate audit command

---

12. Pairing mechanism and Telegram test

Openclaw discourages random users from using your bot by requiring a pairing flow.

1. From Telegram, search for your bot by its username and send any message.
2. The bot will respond with a **pairing code** instead of executing commands.
3. On the VM, in your terminal (as `openclaw`), run the pairing command (check docs; the video shows entering the code in the terminal to approve the user).
4. Once paired, you can chat normally.

Test:

- Ask for the current date/time. Unlike a static LLM, Openclaw knows the host environment and will return the actual VM time and system details, proving shell/environment awareness.
- Ask for a nicely formatted system overview (CPU, memory, disk usage). Openclaw will run the relevant commands and format the output.
- Try asking it to locate the largest files on the server; it will preserve context between messages and refine queries.

Use caution with destructive commands you authorize (e.g. cleaning npm cache); validate suggestions before approval.

13. Security recap

This setup is designed so that even if Openclaw is exploited, the blast radius is contained.

Key decisions:

- **VM isolation** instead of LXC to avoid container-to-host kernel sharing.
- **Dedicated non-root user** for Openclaw; attackers are stuck in a restricted account.
- **Loopback-only gateway**: no inbound ports, only outbound polling of Telegram (or other clients). Telegram servers hold messages until Openclaw polls every few seconds.
- **Pairing process**: even if someone finds your Telegram bot username, they cannot use the agent without an approved pairing.
- **Security audit + manual token fix**: ensures internal tokens and permissions are correctly set.

With this in place, you’re not one of the many users exposing full shell agents unauthenticated on the public internet.

---

14. Ideas for next steps

- Switch from remote APIs to local models (e.g. via Ollama) so the VM uses your own GPU/LLM stack.
- Add skills/plugins gradually for home automation and other workflows; keep auditing configs as you expand.
- Integrate with more chat/voice interfaces while keeping the gateway bound to localhost plus your chosen secure tunnels/VPN.

Ins0mnia