Remote access

Remote Access

This repo supports “remote over SSH” by keeping a single Gateway (the master) running on a dedicated host (desktop/server) and connecting clients to it.

- For **operators (you / the macOS app)**: SSH tunneling is the universal fallback.
- For **nodes (iOS/Android and future devices)**: connect to the Gateway **WebSocket** (LAN/tailnet or SSH tunnel as needed).

The core idea

- The Gateway WebSocket binds to **loopback** on your configured port (defaults to 18789).
- For remote use, you forward that loopback port over SSH (or use a tailnet/VPN and tunnel less).

Common VPN/tailnet setups (where the agent lives)

Think of the **Gateway host** as “where the agent lives.” It owns sessions, auth profiles, channels, and state. Your laptop/desktop (and nodes) connect to that host.

1) Always-on Gateway in your tailnet (VPS or home server)

Run the Gateway on a persistent host and reach it via **Tailscale** or SSH.

- **Best UX:** keep `gateway.bind: "loopback"` and use **Tailscale Serve** for the Control UI.
- **Fallback:** keep loopback + SSH tunnel from any machine that needs access.
- **Examples:** exe.dev (easy VM) or Hetzner (production VPS).

This is ideal when your laptop sleeps often but you want the agent always-on.

2) Home desktop runs the Gateway, laptop is remote control

The laptop does **not** run the agent. It connects remotely:

- Use the macOS app's **Remote over SSH** mode (Settings → General → "OpenClaw runs").
- The app opens and manages the tunnel, so WebChat + health checks "just work."

Runbook: macOS remote access.

3) Laptop runs the Gateway, remote access from other machines

Keep the Gateway local but expose it safely:

- SSH tunnel to the laptop from other machines, or
- Tailscale Serve the Control UI and keep the Gateway loopback-only.

Guide: Tailscale and Web overview.

Command flow (what runs where)

One gateway service owns state + channels. Nodes are peripherals.

Flow example (Telegram → node):

- Telegram message arrives at the **Gateway**.
- Gateway runs the **agent** and decides whether to call a node tool.
- Gateway calls the **node** over the Gateway WebSocket (`node.*` RPC).
- Node returns the result; Gateway replies back out to Telegram.

Notes:

- **Nodes do not run the gateway service.** Only one gateway should run per host unless you intentionally run isolated profiles (see **Multiple gateways**).
- macOS app “node mode” is just a node client over the Gateway WebSocket.

SSH tunnel (CLI + tools)

Create a local tunnel to the remote Gateway WS:

```
ssh -N -L 18789:127.0.0.1:18789 user@host
```

With the tunnel up:

- `openclaw health` and `openclaw status --deep` now reach the remote gateway via `ws://127.0.0.1:18789` .
- `openclaw gateway {status,health,send,agent,call}` can also target the forwarded URL via `--url` when needed.

Note: replace `18789` with your configured `gateway.port` (or `--port` / `OPENCLAW_GATEWAY_PORT`). Note: when you pass `--url` , the CLI does not fall back to config or environment credentials. Include `--token` or `--password` explicitly. Missing explicit credentials is an error.

CLI remote defaults

You can persist a remote target so CLI commands use it by default:

```
{
  gateway: {
    mode: "remote",
    remote: {
      url: "ws://127.0.0.1:18789",
      token: "your-token",
    },
  },
}
```

When the gateway is loopback-only, keep the URL at `ws://127.0.0.1:18789` and open the SSH tunnel first.

Chat UI over SSH

WebChat no longer uses a separate HTTP port. The SwiftUI chat UI connects directly to the Gateway WebSocket.

- Forward `18789` over SSH (see above), then connect clients to `ws://127.0.0.1:18789`.
- On macOS, prefer the app's "Remote over SSH" mode, which manages the tunnel automatically.

macOS app "Remote over SSH"

The macOS menu bar app can drive the same setup end-to-end (remote status checks, WebChat, and Voice Wake forwarding).

Runbook: .

Security rules (remote/VPN)

Short version: **keep the Gateway loopback-only** unless you're sure you need a bind. ›

- **Loopback + SSH/Tailscale Serve** is the safest default (no public exposure).
- **Non-loopback binds** (`lan` / `tailnet` / `custom` , or `auto` when loopback is unavailable) must use auth tokens/passwords.
- `gateway.remote.token` is **only** for remote CLI calls — it does **not** enable local auth.
- `gateway.remote.tlsFingerprint` pins the remote TLS cert when using `wss://` .
- **Tailscale Serve** can authenticate via identity headers when `gateway.auth.allowTailscale: true` . Set it to `false` if you want tokens/passwords instead.
- Treat browser control like operator access: tailnet-only + deliberate node pairing.

Deep dive: **Security**.

‹ Bonjour Discovery Remote Gateway Setup ›