Web interfaces

Control UI

The Control UI is a small **Vite + Lit** single-page app served by the Gateway:

- default: `http://<host>:18789/`
- optional prefix: set `gateway.controlUi.basePath` (e.g. `/openclaw`)

It speaks **directly to the Gateway WebSocket** on the same port.

Quick open (local)

If the Gateway is running on the same computer, open:

- http://127.0.0.1:18789/ (or http://localhost:18789/)

If the page fails to load, start the Gateway first: `openclaw gateway`.

Auth is supplied during the WebSocket handshake via:

- `connect.params.auth.token`
- `connect.params.auth.password`

The dashboard settings panel lets you store a token; passwords are not persisted. The onboarding wizard generates a gateway token by default, so paste it here on first connect.

Device pairing (first connection)

When you connect to the Control UI from a new browser or device, the Gateway requires a **one-time pairing approval** — even if you're on the same Tailnet with `gateway.auth.allowTailscale: true`. This is a security measure to prevent unauthorized access.

What you'll see: "disconnected (1008): pairing required"

To approve the device:

```
# List pending requests
openclaw devices list

# Approve by request ID
openclaw devices approve <requestId>
```

Once approved, the device is remembered and won't require re-approval unless you revoke it with `openclaw devices revoke --device <id> --role <role>`. See for token rotation and revocation.

Notes:

- Local connections (`127.0.0.1`) are auto-approved.
- Remote connections (LAN, Tailnet, etc.) require explicit approval.
- Each browser profile generates a unique device ID, so switching browsers or clearing browser data will require re-pairing.

What it can do (today)

- Chat with the model via Gateway WS (`chat.history` , `chat.send` , `chat.abort` , `chat.inject`)
- Stream tool calls + live tool output cards in Chat (agent events)
- Channels: WhatsApp/Telegram/Discord/Slack + plugin channels (Mattermost, etc.) status + QR login + per-channel config (`channels.status` , `web.login.*` , `config.patch`)

- Instances: presence list + refresh (`system-presence`)
- Sessions: list + per-session thinking/verbose overrides (`sessions.list` , `sessions.patch`)
- Cron jobs: list/add/run/enable/disable + run history (`cron.*`)
- Skills: status, enable/disable, install, API key updates (`skills.*`)
- Nodes: list + caps (`node.list`)
- Exec approvals: edit gateway or node allowlists + ask policy for `exec host=gateway/node` (`exec.approvals.*`)
- Config: view/edit `~/.openclaw/openclaw.json` (`config.get` , `config.set`)
- Config: apply + restart with validation (`config.apply`) and wake the last active session
- Config writes include a base-hash guard to prevent clobbering concurrent edits
- Config schema + form rendering (`config.schema` , including plugin + channel schemas); Raw JSON editor remains available
- Debug: status/health/models snapshots + event log + manual RPC calls (`status` , `health` , `models.list`)
- Logs: live tail of gateway file logs with filter/export (`logs.tail`)
- Update: run a package/git update + restart (`update.run`) with a restart report

Cron jobs panel notes:

- For isolated jobs, delivery defaults to announce summary. You can switch to none if you want internal-only runs.
- Channel/target fields appear when announce is selected.
- Webhook mode uses `delivery.mode = "webhook"` with `delivery.to` set to a valid HTTP(S) webhook URL.
- For main-session jobs, webhook and none delivery modes are available.

Set `cron.webhookToken` to send a dedicated bearer token, if omitted the webhook is sent without an auth header.

Deprecated fallback: stored legacy jobs with `notify: true` can still use `cron.webhook` until migrated.

Chat behavior

`chat.send` is **non-blocking**: it acks immediately with `{ runId, status: "started" }` and the response streams via `chat` events.

Re-sending with the same `idempotencyKey` returns `{ status: "in_flight" }` while running, and `{ status: "ok" }` after completion.

`chat.history` responses are size-bounded for UI safety. When transcript entries are too large, Gateway may truncate long text fields, omit heavy metadata blocks, and replace oversized messages with a placeholder (`[chat.history omitted: message too large]`).

`chat.inject` appends an assistant note to the session transcript and broadcasts a `chat` event for UI-only updates (no agent run, no channel delivery).

Stop:

- Click **Stop** (calls `chat.abort`)
- Type `/stop` (or `stop|esc|abort|wait|exit|interrupt`) to abort out-of-band
- `chat.abort` supports `{ sessionKey }` (no `runId`) to abort all active runs for that session

Abort partial retention:

- When a run is aborted, partial assistant text can still be shown in the UI
- Gateway persists aborted partial assistant text into transcript history when buffered output exists

Persisted entries include abort metadata so transcript consumers can tell abort partials from normal completion output

›

Tailnet access (recommended)

Integrated Tailscale Serve (preferred)

Keep the Gateway on loopback and let Tailscale Serve proxy it with HTTPS:

```
openclaw gateway --tailscale serve
```

Open:

`https://<magicdns>/` (or your configured `gateway.controlUi.basePath`)

By default, Serve requests can authenticate via Tailscale identity headers (`tailscale-user-login`) when `gateway.auth.allowTailscale` is `true`. OpenClaw verifies the identity by resolving the `x-forwarded-for` address with `tailscale whois` and matching it to the header, and only accepts these when the request hits loopback with Tailscale's `x-forwarded-*` headers. Set `gateway.auth.allowTailscale: false` (or force `gateway.auth.mode: "password"`) if you want to require a token/password even for Serve traffic.

Bind to tailnet + token

```
openclaw gateway --bind tailnet --token "$(openssl rand -hex 32)"
```

Then open:

`http://<tailscale-ip>:18789/` (or your configured `gateway.controlUi.basePath`)

Paste the token into the UI settings (sent as `connect.params.auth.token`).

Insecure HTTP

If you open the dashboard over plain HTTP (`http://<lan-ip>` or `http://<tailscale-ip>`), the browser runs in a **non-secure context** and blocks WebCrypto. By default, OpenClaw **blocks** Control UI connections without device identity.

Recommended fix: use HTTPS (Tailscale Serve) or open the UI locally:

- `https://<magicdns>/` (Serve)
- `http://127.0.0.1:18789/` (on the gateway host)

Downgrade example (token-only over HTTP):

```
{
  gateway: {
    controlUi: { allowInsecureAuth: true },
    bind: "tailnet",
    auth: { mode: "token", token: "replace-me" },
  },
}
```

This disables device identity + pairing for the Control UI (even on HTTPS). Use only if you trust the network.

See for HTTPS setup guidance.

Building the UI

The Gateway serves static files from `dist/control-ui`. Build them with:

```
pnpm ui:build # auto-installs UI deps on first run
```

Optional absolute base (when you want fixed asset URLs):

```
OPENCLAW_CONTROL_UI_BASE_PATH=/openclaw/ pnpm ui:build
                              ›
```

For local development (separate dev server):

```
pnpm ui:dev # auto-installs UI deps on first run
```

Then point the UI at your Gateway WS URL (e.g. `ws://127.0.0.1:18789`).

Debugging/testing: dev server + remote Gateway

The Control UI is static files; the WebSocket target is configurable and can be different from the HTTP origin. This is handy when you want the Vite dev server locally but the Gateway runs elsewhere.

1. Start the UI dev server: `pnpm ui:dev`
2. Open a URL like:

```
http://localhost:5173/?gatewayUrl=ws://<gateway-host>:18789
```

Optional one-time auth (if needed):

```
http://localhost:5173/?gatewayUrl=wss://<gateway-host>:18789&token=<ga
```

Notes:

- `gatewayUrl` is stored in localStorage after load and removed from the URL.
- `token` is stored in localStorage; `password` is kept in memory only.

When `gatewayUrl` is set, the UI does not fall back to config or environment credentials. Provide `token` (or `password`) explicitly. Missing explicit credentials is an error.

Use `wss://` when the Gateway is behind TLS (Tailscale Serve, HTTPS proxy, etc.).

`gatewayUrl` is only accepted in a top-level window (not embedded) to prevent clickjacking.

For cross-origin dev setups (e.g. `pnpm ui:dev` to a remote Gateway), add the UI origin to `gateway.controlUi.allowedOrigins`.

Example:

```
{
  gateway: {
    controlUi: {
      allowedOrigins: ["http://localhost:5173"],
    },
  },
}
```

Remote access setup details: .

‹ Web Dashboard ›