

cs-proxy

Manual

HTTP/HTTPS edge for napp-it 4ai with static file serving, TLS and an optional reverse proxy for local AI backends

Version 0.12.1 · October 13, 2026

Contents

1. Overview	3
1.1 Why	3
1.2 Architecture	3
2. Configuration	3
3. Standalone and universal mode	4
4. AI edge (local LLM reverse proxy)	4
5. Console output	5
6. Integration (start.pl / monitor.pl)	5
7. Build	5
8. License	5

1. Overview

cs-proxy is the HTTP/HTTPS edge of napp-it 4ai (client/server web GUI). It serves the static document root directly (gzip, ETag/304, in-memory LRU cache), terminates TLS and reverse-proxies the dynamic requests to a persistent Perl worker (`webserver.pl -worker`, loopback only) or to the legacy webserver.

It ships inside the napp-it distribution at `data/cs_server/proxy/<platform>.<arch>/cs-proxy[.exe]` - the correct binary for the host OS is always present, no download is needed.

Optionally cs-proxy also acts as a reverse proxy for local AI backends (llama-server, Ollama) with API key and IP protection, and it can run as a generic reverse proxy without any napp-it coupling.

1.1 Why

The bundled Perl has no `Net::SSL` / `IO::Socket::SSL` XS extension, so the TLS of the webserver bootstraps against the system Perl - a version mismatch on every OS (development system versus member). Moving TLS into a static Go binary removes this fragility completely: the Perl worker runs plain HTTP on loopback only.

1.2 Architecture

```
browser --HTTPS/HTTP--> cs-proxy (Go, ports 80/443)
    |- static: serves data/wwwroot directly (cache/gzip/ETag)
    |- dynamic: reverse-proxies /cgi-bin/* (SSE/upload streamed)
    '- /ping, /healthz for the monitor.pl watchdog
webserver.pl -worker <-- loopback 127.0.0.1:8001 (dynamic only, no TLS, no static)

AI clients --HTTPS/HTTP--> cs-proxy ai edge (optional, ai_listen_http/https)
    '- reverse-proxies to the local LLM backend
      (llama-server / Ollama, ai_upstream)
```

2. Configuration

There is **no separate proxy config**: the proxy settings live in the same file the webserver uses, `_cfg/webserver/webserver.conf` (flat `key = value`, `#` comments).

```
# --- webserver.pl (plain HTTP) ---
http_port    = 800          # HTTP only; the edge owns 80/443
remote_http  = proxy        # deny | allow | proxy
listen_addr  = 0.0.0.0
default_url  = /cgi-bin/admin.pl

# --- cs-proxy edge (Go; same file, proxy_* keys) ---
proxy = on                  # off disables the edge (webserver.pl serves directly)
proxy_listen_addr = 0.0.0.0
proxy_listen_http = 80
proxy_listen_https = 443
proxy_cache = on
proxy_cache_max_mb = 64
proxy_cache_ttl_s = 60
proxy_compress = gzip

# --- optional AI edge (local LLM reverse proxy) ---
ai = off                    # off disables the AI listener
ai_listen_addr = 0.0.0.0
ai_listen_http = 0          # 0 = no plain HTTP (recommended)
ai_listen_https = 8443
ai_upstream = http://127.0.0.1:8080      # default backend (no prefix)
ai_upstream_ollama = http://127.0.0.1:11434 # named backend -> /ollama/...
ai_keys_file = _cfg/cs-proxy.keys        # API keys, one per line; hot-reloaded (sent as Bearer)
ai_allowed_ip = 192.168.2.0/24,10.0.0.5  # optional IP/CIDR allowlist
ai_upstream_key =                    # optional: API key injected to backends (as Bearer; edge key stripped)
ai_cert =                    # optional certificate for the AI HTTPS listener
```

```
ai_key_file =                # optional key (else the shared cs-proxy-cert.pem)

# --- optional web GUI hardening ---
proxy_allowed_ip = 192.168.2.0/24  # GUI edge IP/CIDR allowlist (remote only; loopback and /ping
    always allowed)
```

- ****The upstream port is derived from `http_port`**** - the proxy always follows the actual port of `webserver.pl`; `proxy_upstream` overrides it.
- **Migration:** an old `http_port = 80` (or 8080) in `webserver.conf` is rewritten to 800 automatically at startup.
- **remote_http** (`webserver.pl`): `deny` blocks remote HTTP (403), `allow` serves it directly, `proxy` (the default) forwards remote HTTP to https at the `/` landing page.
- A legacy standalone `_cfg/cs-proxy.cfg` still works via `-config`.
- One instance can run both edges, the web GUI (80/443) and the AI listener (its own ports), side by side. `proxy=off` plus `ai=on` gives a dedicated AI server.

3. Standalone and universal mode

cs-proxy can run as a generic reverse proxy without any napp-it coupling:

```
cs-proxy --forward http://127.0.0.1:9000 --http 8080 --https 8443 --addr 0.0.0.0
```

Switches: `-conf / -config <file>`, `-forward <url>` (upstream override), `-http <port>`, `-https <port>`, `-addr <host>`.

A dedicated AI server is just as simple (`--ai-forward` alone disables the web edge; combine both flags to run both edges):

```
cs-proxy --ai-forward http://127.0.0.1:8080 --ai-http 11434 --ai-https 8443 \
    --ai-addr 0.0.0.0 --ai-keys-file _cfg/cs-proxy.keys \
    --allowed-ai 192.168.2.0/24 --allowed-proxy 192.168.2.0/24
```

AI switches: `-ai-forward <url>` (default upstream and enables the AI listener), `-ai-http <port>`, `-ai-https <port>`, `-ai-addr <host>`, `-ai-keys-file <file>`, `--allowed-ai <ip/cidr,...>` (alias `-ai-allowed-ip`), `--allowed-proxy <list>`.

4. AI edge (local LLM reverse proxy)

The AI edge serves OpenAI-compatible LLM backends (llama-server `/v1/*`, Ollama `/api/*`) through cs-proxy on its own listener, so external clients never touch the raw backend port.

- **Backend selection by path prefix:** `ai_upstream` is the default backend (no prefix); every `ai_upstream_<name>` is routed under `/<name>/`. `https://host:8443/ollama/v1/chat/completions` reaches Ollama, while `https://host:8443/v1/chat/completions` reaches the default (llama-server). Both can run side by side; the client just picks the prefix.
- **Key list:** `ai_keys_file` (`_cfg/cs-proxy.keys`) holds the API keys - one per line, `#` comments, optional `key = label` (the label is display-only). Keys are **hot-reloaded on change**, so adding or deleting a key in the GUI works without a restart. `ai_key` is deprecated (still accepted, prints a warning).
- **Authentication only on the AI listener:** the web GUI listener keeps its existing behavior (no cookie/API-key conflict). `ai_allowed_ip` restricts the client IPs/CIDRs; both mechanisms are optional and can be combined.
- **Auth strip and injection:** the `Authorization` header of the client never reaches the backend (it is removed before the upstream hop, so the edge key stays out of the LLM logs). If a backend needs its own key, `ai_upstream_key` re-injects `Authorization: Bearer <ai_upstream_key>` to every backend.
- ****Aggregated `/v1/models`**** with more than one backend, `GET /v1/models` on the AI listener merges the model lists of all backends and prefixes the ids with the backend name (`ollama/llama3.1`), plus `x_csproxy_backend` and `x_csproxy_endpoint` hints for a GUI model dropdown. The list is cached for 10 s.

- **Streaming works out of the box:** `FlushInterval = -1` passes SSE and token chunks through without buffering.
- **Certificate:** the AI HTTPS listener uses `ai_cert` / `ai_key_file` when set, otherwise the shared generated `_cfg/cs-proxy-cert.pem` (the same as the GUI edge).
- **Web GUI hardening:** `proxy_allowed_ip` blocks remote GUI clients outside the IP/CIDR list with 403 (loopback and `/ping` are always allowed, so local administration and the monitor.pl watchdog keep working) - for example for dual-NIC setups where the AI listener is public but the web GUI is internal only.
- **Logging:** requests are logged with the note column `ai`, for example `HTTP 200 POST /v1/chat/completions 10.0.0.5 8123ms ai` (rejections too).

5. Console output

cs-proxy logs every request to its console (stdout/stderr) in the same format webserver.pl uses - useful when it runs via `start.pl 4` (one minimized window per component):

```
2026/08/29 11:16:26 HTTP 200 GET /cgi-bin/admin.pl 127.0.0.1 1633ms proxy
2026/08/29 11:16:26 HTTP 200 GET /_doc/menu/ico/131.png 127.0.0.1 174ms static
2026/08/29 11:16:26 HTTP 302 GET / 127.0.0.1 0ms root
2026/08/29 11:16:26 HTTP 200 POST /cgi-bin/get_async.pl 127.0.0.1 118ms proxy
```

The note column is `ping`, `root`, `proxy` (upstream), `static` (served from the docroot) or `ai` (AI edge). The startup lines (`docroot=...` `upstream=...` `HTTP/HTTPS listening`, `ai edge upstream=...`) and TLS handshake errors go to the same console.

6. Integration (start.pl / monitor.pl)

- `start.pl 9` detects the proxy mode via `_proxy_bin()` and starts `webserver.pl -worker` plus `cs-proxy -config ...`; the options 0, 1 and 2 stop both.
- `monitor.pl` supervises BOTH services in proxy mode (cs-proxy on port 80 `/ping`, the worker on `worker_port` 8001) and restarts each with its proper arguments.
- The `webserver.conf` key `worker_port` sets the loopback port of the worker (default 8001; the legacy ds-proxy example already occupies 8000).

7. Build

Go 1.22 or newer is required.

```
CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build -trimpath -ldflags "-s -w" -o cs-proxy .
```

The CI workflow [.github/workflows/release.yml](#) builds all 8 platforms (mswin, linux, illumos, solaris, freebsd, darwin x amd64/arm64) into the layout used by the distribution (`data/cs_server/proxy/<platform>.<arch>/`) and attaches the tarballs to the matching GitHub release (tag `v*`).

8. License

BSD 2-Clause - see the file LICENSE. Copyright (c) 2026 Guenther Alka / napp-it.org.