

cs-proxy

Handbuch

HTTP/HTTPS-Edge für napp-it 4ai mit statischer Auslieferung, TLS und optionalem Reverse-Proxy für lokale KI-Backends

Version 0.12.1 · 13. Oktober 2026

Inhalt

1. Überblick	3
1.1 Warum	3
1.2 Architektur	3
2. Konfiguration	3
3. Eigenständiger und universeller Modus	4
4. AI-Edge (lokaler LLM-Reverse-Proxy)	4
5. Konsolenausgabe	5
6. Einbindung (start.pl / monitor.pl)	5
7. Bauen	5
8. Lizenz	5

1. Überblick

cs-proxy ist der HTTP/HTTPS-Edge von napp-it 4ai (Client/Server Web-GUI). Er liefert das statische Document-Root direkt aus (gzip, ETag/304, In-Memory-LRU-Cache), terminiert TLS und leitet die dynamischen Anfragen per Reverse-Proxy an einen dauerhaft laufenden Perl-Worker (`webserver.pl -worker`, nur Loopback) oder an den klassischen Webserver weiter.

Er ist Teil der napp-it-Distribution unter `data/cs_server/proxy/<platform>.<arch>/cs-proxy[.exe]` - das passende Binary für das Host-Betriebssystem ist immer vorhanden, ein Download ist nicht nötig.

Optional dient cs-proxy auch als Reverse-Proxy für lokale KI-Backends (llama-server, Ollama) mit API-Key- und IP-Absicherung, und er kann als allgemeiner Reverse-Proxy ganz ohne napp-it-Bezug laufen.

1.1 Warum

Das mitgelieferte Perl hat keine XS-Erweiterung `Net::SSL` / `IO::Socket::SSL`, sodass das TLS des Webserverns gegen das System-Perl aufgebaut wird - auf jedem Betriebssystem eine Versionsabweichung (Entwicklungssystem gegen Member). Das TLS in ein statisches Go-Binary zu verlegen, beseitigt diese Fragilität vollständig: Der Perl-Worker spricht nur noch einfaches HTTP auf Loopback.

1.2 Architektur

```
Browser --HTTPS/HTTP--> cs-proxy (Go, Ports 80/443)
                        |- statisch: liefert data/wwwroot direkt (Cache/gzip/ETag)
                        |- dynamisch: Reverse-Proxy für /cgi-bin/* (SSE/Upload gestreamt)
                        '- /ping, /healthz für den Watchdog monitor.pl

webserver.pl -worker <-- Loopback 127.0.0.1:8001 (nur dynamisch, kein TLS, nichts Statisches)

KI-Clients --HTTPS/HTTP--> cs-proxy AI-Edge (optional, ai_listen_http/https)
                        '- Reverse-Proxy zum lokalen LLM-Backend
                          (llama-server / Ollama, ai_upstream)
```

2. Konfiguration

Es gibt **keine eigene Proxy-Konfiguration**: Die Proxy-Einstellungen stehen in derselben Datei wie die des Webserverns, `_cfg/webserver/webserver.conf` (flach `schlüssel = wert`, `#` für Kommentare).

```
# --- webserver.pl (einfaches HTTP) ---
http_port      = 800                # nur HTTP; der Edge besitzt 80/443
remote_http    = proxy              # deny | allow | proxy
listen_addr    = 0.0.0.0
default_url    = /cgi-bin/admin.pl

# --- cs-proxy Edge (Go; gleiche Datei, proxy_*-Schlüssel) ---
proxy          = on                 # off schaltet den Edge ab (webserver.pl liefert direkt aus)
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

# --- optionaler AI-Edge (lokaler LLM-Reverse-Proxy) ---
ai = off                            # off schaltet den AI-Listener ab
ai_listen_addr = 0.0.0.0
ai_listen_http = 0                  # 0 = kein einfaches HTTP (empfohlen)
ai_listen_https = 8443
ai_upstream     = http://127.0.0.1:8080      # Standard-Backend (kein Präfix)
ai_upstream_ollama = http://127.0.0.1:11434  # benanntes Backend -> /ollama/...
ai_keys_file    = _cfg/cs-proxy.keys        # API-Keys, einer pro Zeile; Hot-Reload (als Bearer
gesendet)
ai_allowed_ip    = 192.168.2.0/24,10.0.0.5    # optionale IP/CIDR-Positivliste
```

```

ai_upstream_key =          # optional: API-Key, der an die Backends übergeben wird (als Bearer;
    Edge-Key entfernt)
ai_cert =                 # optionales Zertifikat für den AI-HTTPS-Listener
ai_key_file =             # optionaler Schlüssel (sonst das gemeinsame cs-proxy-cert.pem)

# --- optionale Härtung der Web-GUI ---
proxy_allowed_ip = 192.168.2.0/24 # IP/CIDR-Positivliste des GUI-Edge (nur remote; Loopback und
    /ping immer erlaubt)

```

- ****Der Upstream-Port wird aus `http_port` abgeleitet**** - der Proxy folgt immer dem tatsächlichen Port von `webserver.pl`; `proxy_upstream` überschreibt das.
- **Migration:** Ein alter Eintrag `http_port = 80` (oder 8080) in `webserver.conf` wird beim Start automatisch in `800` umgeschrieben.
- **remote_http** (`webserver.pl`): `deny` blockiert Remote-HTTP (403), `allow` liefert es direkt aus, `proxy` (Standard) leitet Remote-HTTP auf der Startseite / auf https um.
- Eine alte, eigenständige `_cfg/cs-proxy.cfg` funktioniert weiter über `-config`.
- Eine Instanz kann beide Edges nebeneinander betreiben, die Web-GUI (80/443) und den AI-Listener (eigene Ports). `proxy=off` plus `ai=on` ergibt einen reinen KI-Server.

3. Eigenständiger und universeller Modus

cs-proxy kann als allgemeiner Reverse-Proxy ganz ohne napp-it-Bezug laufen:

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

Schalter: `-conf / -config <datei>`, `-forward <url>` (Upstream überschreiben), `-http <port>`, `-https <port>`, `-addr <host>`.

Ein reiner KI-Server ist genauso einfach (`--ai-forward` allein schaltet den Web-Edge ab; beide Schalter kombiniert betreiben beide 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

```

KI-Schalter: `-ai-forward <url>` (Standard-Upstream und aktiviert den AI-Listener), `-ai-http <port>`, `-ai-https <port>`, `-ai-addr <host>`, `-ai-keys-file <datei>`, `-allowed-ai <ip/cidr,...>` (Alias `-ai-allowed-ip`), `-allowed-proxy <liste>`.

4. AI-Edge (lokaler LLM-Reverse-Proxy)

Der AI-Edge liefert OpenAI-kompatible LLM-Backends (llama-server `/v1/*`, Ollama `/api/*`) über cs-proxy auf einem eigenen Listener aus, sodass externe Clients nie den rohen Backend-Port berühren.

- **Backend-Auswahl per Pfad-Präfix:** `ai_upstream` ist das Standard-Backend (ohne Präfix); jedes `ai_upstream_<name>` wird unter `/<name>/` geroutet. `https://host:8443/ollama/v1/chat/completions` erreicht Ollama, während `https://host:8443/v1/chat/completions` das Standard-Backend (llama-server) erreicht. Beide können nebeneinander laufen; der Client wählt nur das Präfix.
- **Key-Liste:** `ai_keys_file` (`_cfg/cs-proxy.keys`) enthält die API-Keys - einer pro Zeile, `#` für Kommentare, optional `key = label` (das Label dient nur der Anzeige). Keys werden **bei Änderung neu geladen**, Hinzufügen und Löschen in der GUI funktioniert also ohne Neustart. `ai_key` ist veraltet (wird noch akzeptiert und gibt eine Warnung aus).
- **Authentifizierung nur am AI-Listener:** Der Web-GUI-Listener behält sein bisheriges Verhalten (kein Konflikt zwischen Cookie und API-Key). `ai_allowed_ip` beschränkt die Client-IPs/CIDRs; beide Mechanismen sind optional und kombinierbar.
- **Entfernen und Einsetzen der Authentifizierung:** Der `Authorization`-Header des Clients erreicht das Backend nie (er wird vor dem Upstream-Schritt entfernt, sodass der Edge-Key aus den LLM-Logs herausbleibt). Braucht ein Backend einen eigenen Key, setzt `ai_upstream_key` für jedes Backend `Authorization: Bearer <ai_upstream_key>` neu ein.

- ****Zusammengeführtes /v1/models:**** Bei mehr als einem Backend führt `GET /v1/models` am AI-Listener die Modelllisten aller Backends zusammen und stellt den IDs den Backend-Namen voran (`ollama/llama3.1`), dazu die Hinweise `x_csproxy_backend` und `x_csproxy_endpoint` für ein Modell-Dropdown der GUI. Die Liste wird 10 s gecacht.
- **Streaming funktioniert direkt:** `FlushInterval = -1` reicht SSE- und Token-Blöcke ohne Pufferung durch.
- **Zertifikat:** Der AI-HTTPS-Listener nutzt `ai_cert` / `ai_key_file`, wenn gesetzt, sonst das gemeinsame erzeugte `_cfg/cs-proxy-cert.pem` (wie der GUI-Edge).
- **Härtung der Web-GUI:** `proxy_allowed_ip` blockiert Remote-GUI-Clients außerhalb der IP/CIDR-Liste mit 403 (Loopback und `/ping` sind immer erlaubt, lokale Administration und der Watchdog `monitor.pl` funktionieren also weiter) - zum Beispiel bei Dual-NIC-Aufbauten, bei denen der AI-Listener öffentlich, die Web-GUI aber nur intern erreichbar ist.
- **Logging:** Anfragen werden mit der Notizspalte `ai` protokolliert, zum Beispiel `HTTP 200 POST /v1/chat/completions 10.0.0.5 8123ms ai` (Ablehnungen ebenfalls).

5. Konsolenausgabe

cs-proxy protokolliert jede Anfrage auf seiner Konsole (stdout/stderr) im selben Format wie `webserver.pl` - nützlich, wenn er über `start.pl 4` läuft (ein minimiertes Fenster pro Komponente):

```
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
```

Die Notizspalte ist `ping`, `root`, `proxy` (Upstream), `static` (aus dem Docroot ausgeliefert) oder `ai` (AI-Edge). Die Startzeilen (`docroot=... upstream=... HTTP/HTTPS listening, ai edge upstream=...`) und TLS-Handshake-Fehler erscheinen auf derselben Konsole.

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

- `start.pl 9` erkennt den Proxy-Modus über `_proxy_bin()` und startet `webserver.pl -worker` plus `cs-proxy -config ...`; die Optionen 0, 1 und 2 stoppen beides.
- `monitor.pl` überwacht im Proxy-Modus BEIDE Dienste (cs-proxy über `/ping` auf Port 80, den Worker auf `worker_port` 8001) und startet jeden mit seinen passenden Argumenten neu.
- Der Schlüssel `worker_port` in `webserver.conf` setzt den Loopback-Port des Workers (Standard 8001; das alte ds-proxy-Beispiel belegt bereits 8000).

7. Bauen

Erforderlich ist Go 1.22 oder neuer.

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

Der CI-Workflow `.github/workflows/release.yml` baut alle 8 Plattformen (mswin, linux, illumos, solaris, freebsd, darwin x amd64/arm64) in das Layout der Distribution (`data/cs_server/proxy/<platform>.<arch>/`) und hängt die Tarballs an das passende GitHub-Release (Tag `v*`).

8. Lizenz

BSD 2-Clause - siehe Datei LICENSE. Copyright (c) 2026 Guenther Alka / napp-it.org.