CERTYX



AUTONOMOUS CERTIFICATE LIFECYCLE MANAGEMENT

Agent installation

CLM network agent — installation and discovery

LANGUAGE	REVISION	DATE
English	8550060f	1 November 2026

Contents

1 What it does 3
2 Before installation 3
3 Agent address and local test server 3
4 Five steps 3
5 Package verification 4
6 Package downloads 4
7 Linux with systemd 5
8 Windows with Task Scheduler 10
9 Manual / container execution 12
10 Coverage and progress 13
11 Network map 14
12 For the customer's security team 14
13 Operational settings 15
14 Revocation and recovery 16
15 Package production 16
16 SSH 17
17 Customer-controlled secrets and KRL subscriptions 18
18 SSH workload user identities 18
19 TLS client certificate destinations 18
20 Deployment connectors and SSH relay 19
21 Saved lifecycle result delivery 22
22 Protected configuration reads 22
23 Independent private-network verification 22
24 Public CAs: DNS-01, EAB and DigiCert 24
25 Signed updates 24
26 Command-line checks 25
27 Windows host prerequisites 26

1 What it does

The network agent runs inside a customer network. It connects outward to the CLM HTTPS endpoint; CLM does not need incoming access to private subnets. A registered agent reports the private IPv4 networks its host is connected to or routes to (the network map), retrieves its approved discovery profile, finds live hosts in the approved ranges, connects to their TLS ports with a TCP connection plus a direct TLS handshake and uploads public certificates into inventory. It never uploads private keys. Scanning is transparent: it is agreed with the customer's security team in advance (see "For the customer's security team") and never tries to avoid or confuse intrusion detection; pacing, per-host limits, windows and back-off only protect your network and hosts from load. Use one agent per reachable network zone. Installing a discovery agent does not authorize certificate deployment.

2 Before installation

Choose a supported Linux host with systemd or a Windows host with Task Scheduler and network reachability to the authorized targets. Use the package matching the operating system and architecture: the console's commands use `clm-agent-linux-x64.tar.gz` and `clm-agent-win-x64.zip`; install-linux.sh also accepts aarch64 hosts with `clm-agent-linux-arm64.tar.gz` when your operator publishes that package (replace `x64` with `arm64` in the commands). Packages contain the .NET runtime; SDK installation is not needed on target hosts. HTTPS must have a hostname and CA chain trusted by the host OS. The agent address must be HTTPS, reachable from the agent host and use a certificate the agent host trusts and can check for revocation online; a plain-HTTP console on 127.0.0.1 cannot serve agents (see "Agent address and local test server"). Allow outbound HTTPS to CLM and outbound TCP to target CIDRs; no new inbound agent listener is opened.

3 Agent address and local test server

Agents connect to one HTTPS address, the "Address for agents" shown after Create enrollment code. It must meet four conditions, which the installer checks before it uses the code (`Clm.Agent check`): the agent host resolves the name and reaches the port; the server certificate names exactly that host name or IP address and chains to a root in the host's system trust store; the certificate names a CRL distribution point (or OCSP responder) the agent host can download over HTTP; and the clocks differ by less than four minutes. The operator configures it with `Clm:Ingress:AgentHosts` (first entry), optionally `Clm:Ingress:AgentPort` when agents use another port than the console, and `Clm:Ingress:AgentTrustRootFile` when the certificate comes from a private CA: the console then publishes that root at `/downloads/clm-agent-trust-root.crt` and shows the commands to verify its SHA-256 checksum and trust it (see CONFIGURATION.md). Without an agent host, the console offers the HTTPS address it was opened with; opened through a proxy (such as the edge of the container deployment), through which agents cannot authenticate, it offers that host name on `Clm:Ingress:AgentPort` (8443 in the container deployment, which passes TLS through to the agent listener), and without an agent port it offers none; opened over plain HTTP it offers none. In both cases it explains why.

Local test server. `./start.sh` serves the console only on this computer (https://127.0.0.1:7243 with the development certificate, http://127.0.0.1:5180 redirecting to it), which agents on other machines can neither reach nor trust. Start it with the address of this computer that the agent host reaches (its LAN address, a DNS name, or 172.17.0.1 for Docker containers on the same computer):

```
ip -4 addr          # or: hostname -I  - pick the address the agent host reaches
CLM_AGENT_HOST=192.168.1.20 ./start.sh
```

start.sh then creates a local root CA and an HTTPS certificate for that address in `.local-agent-endpoint/` (owner-only, never committed; the CA key stays there), adds an HTTPS listener for agents on port 5443 (`CLM_AGENT_PORT`), publishes the revocation list over HTTP on port 5181 (`CLM_AGENT_CRL_PORT`) and configures the agent host, port and trust root. It prints the agent address; the console offers it with the trust step. The address is remembered for the next `./start.sh`; `CLM_AGENT_HOST=off ./start.sh` disables it. Allow inbound TCP 5443 and 5181 in the firewall of this computer (for example `sudo ufw allow 5443/tcp` and `sudo ufw allow 5181/tcp`). The console itself stays reachable only on 127.0.0.1 and localhost. This is for testing; production uses a host name with a certificate from your PKI or a public CA.

4 Five steps

1. In Agents and destinations, enter a unique agent ID and select Create enrollment code. The console then shows the address for agents, your organization ID and numbered installation steps with ready-to-copy commands for Linux and Windows. Download the enrollment token file and copy it into an empty working directory on the server before anything else: the package archive is served only with a valid enrollment code, so the package download sends the code from that file. It expires in ten minutes and can be used once; checking it during the download does not use it up, enrollment does. Treat it like a password; never pass its contents as a command argument.
2. In that working directory, download the package and check its SHA-256 checksum (step 3 of the Linux or Windows section). When your operator publishes a release signing key, also verify the signed release: download `clm-agent-<rid>.release.json`, `clm-agent-<rid>.release.json.sig` and `verify-release.sh` from the same `/downloads/` address (`<rid>` is `linux-x64` or `win-x64`; a separate agent address serves these files, the two package verifiers and `verify-release.sh` as well), obtain release-signing.pub and its SHA-256 fingerprint from a channel you trust (not only from

the same server), and run `bash verify-release.sh release-signing.pub clm-agent-linux-x64.release.json clm-agent-linux-x64.tar.gz` before extracting; do not install when it fails. Without a published key there is no `.sig` file and this signature check does not apply; such releases cannot be installed as signed updates. Then verify its publisher signature against the publisher certificate fingerprint you obtained from your CLM provider through an independent channel (see "Package verification"); do not install a package that is refused. Windows packages additionally carry Authenticode signatures when the release is signed.

3. Install the package with the commands the console shows (step by step in the Linux and Windows sections below). The agent generates an RSA 3072 client identity locally. CLM stores its public fingerprint in the correct organization. The secret PFX and password file remain on the host.
4. In CLM, select the registered agent and approve its discovery profile: IPv4 CIDRs (the agent's network map helps; public ranges are never suggested and must be typed and acknowledged by the administrator), exclusions, mode (`standard`: liveness and common ports; `full`: additionally every port 1–65535 on live hosts), liveness ports, common ports, rate, connections per host and scan windows with their time zone. Enable the profile. The agent checks every profile itself before it connects: at most 64 CIDRs, each at least /16 and at most 65,536 addresses, at most 262,144 addresses per profile and 256 exclusions (or the stricter limits configured on the server), only well-formed IPv4 CIDRs and exclusions without host bits, never loopback, link-local, multicast or reserved space; public ranges, ranges shorter than /20 and full mode only with the administrator's acknowledgement. A refused profile is reported as invalid and nothing is scanned. The local `AllowedCidrs` list (initially the directly connected IPv4 networks) is a hard boundary only when the host owner sets `RestrictToAllowedCidrs` to `true`.
5. Check the result: the installer ends with `sudo clm-agent status` ("Result: connected"), the console shows "first contact received" for the new agent and its first and last contact under Registered agents. Later, check Last contact, the agent log and newly discovered inventory certificates. Discovered owners start as Unassigned. TLS errors are retained; discovery is not proof that the service certificate is trusted.

5 Package verification

Every release package carries two independent signatures:

- Publisher code signing (checked before installation): `<archive>.sha256` is the release hash and `<archive>.sha256.p7s` its detached CMS signature by the publisher's code-signing certificate (an optional `.p7s.tsr` file carries an RFC 3161 time stamp). Inside the package, `SHA256SUMS` lists the SHA-256 hash of every file and `SHA256SUMS.p7s` signs it; missing, added or changed files and links are refused. Windows packages additionally carry Authenticode signatures on `Clm.Agent.exe`, the `Clm.*.dll` assemblies and the PowerShell scripts.
- The offline ECDSA P-384 release manifest (`.release.json` and `.sig`) used by `verify-release.sh` and by signed Linux updates (see "Signed updates").

The trust anchor for code signing is the SHA-256 fingerprint of the publisher certificate. Do not take it from the download server: obtain it from your CLM provider through an independent channel (contract documentation, support portal or release notes) and keep it pinned. Optionally also require a chain to the publisher's CA with `--ca-file` / `-CaFile`. Verification works offline and needs only bash, OpenSSL 3 and coreutils (Linux) or Windows PowerShell 5.1 / PowerShell 7 (Windows).

```
bash verify-agent-package.sh --publisher-sha256 FINGERPRINT clm-agent-linux-x64.tar.gz
```

```
Get-AuthenticodeSignature .\verify-agent-package.ps1
.\verify-agent-package.ps1 -PublisherSha256 FINGERPRINT -Archive .\clm-agent-win-x64.zip
```

`VERIFIED` with exit code 0 means the signature, the pinned publisher, the code-signing usage, the certificate validity and every hash match. `REFUSED: <reason>` with exit code 1 means do not install: a missing signature, an unexpected publisher, a hash mismatch or an expired certificate. Download the package again and contact your provider if the refusal persists. With `--tsa-ca-file` (Linux verifier only), the RFC 3161 time stamp is required and the certificate validity is checked at the signing time. `--allow-unsigned-development` / `-AllowUnsignedDevelopment` exists only for development builds: it accepts a missing signature after hash checks and prints a loud warning. Never use it on production hosts. The console builds its installation commands from the package catalog: when the server publishes a package without a publisher signature (no `.sha256.p7s`, an unsigned development build), the commands say so in a comment line, skip the `.p7s` download, check the checksum and use these development overrides; a signed package always gets the pinned-publisher commands. Both verifiers also check an extracted package with `--directory <dir>` / `-Directory <dir>` (signed `SHA256SUMS`, every hash and the exact file set); this is what the installers run. In that mode on Windows the PowerShell verifier additionally checks the Authenticode signatures of `Clm.Agent.exe`, the `Clm.*.dll` assemblies and the PowerShell scripts against the same pinned publisher (the chain must be trusted on the host, or pass `-CaFile`); on Linux and macOS it prints a note and relies on the signed manifest. Checking an archive (`-Archive`) verifies the signed archive hash only.

The installers repeat the verification and fail closed before changing the host. Because that check runs code from the package itself, it protects against corrupted or partially modified packages and enforces your pin; it does not replace verifying the archive with the separately downloaded verifier first.

6 Package downloads

The package archives (`/downloads/clm-agent-*.tar.gz` and `.zip`) contain executable code and are served only to a request that presents one of:

- a signed-in console session: the package cards in Agents and destinations and in SSH automation work in your browser. Each card shows the archive's file name, version, size, SHA-256 (with a Copy button), whether a publisher signature (`.p7s`) and a signed release manifest (`.sig`) are published, or that the package is not published on this server; its buttons download the package, its `.sha256` , `.p7s` and `.tsr` files, the release manifest and its signature and the verifier, each saved under its own file name (the server sends every file under `/downloads/` as an attachment). The card reports which signature files exist; the verifier on the host checks them;
- a valid, unused and unexpired enrollment code of your organization, sent as the headers `X-CLM-Enrollment-Token` and `X-CLM-Tenant` (the organization ID). The console's commands read the code from clm-enrollment.token in the working directory; on Linux it is piped to curl and never appears on a command line. Checking the code does not use it up; it is used once, by enrollment, and the ten-minute validity covers download and installation together;
- an enrolled agent's TLS client certificate with its organization (signed updates, see "Signed updates").

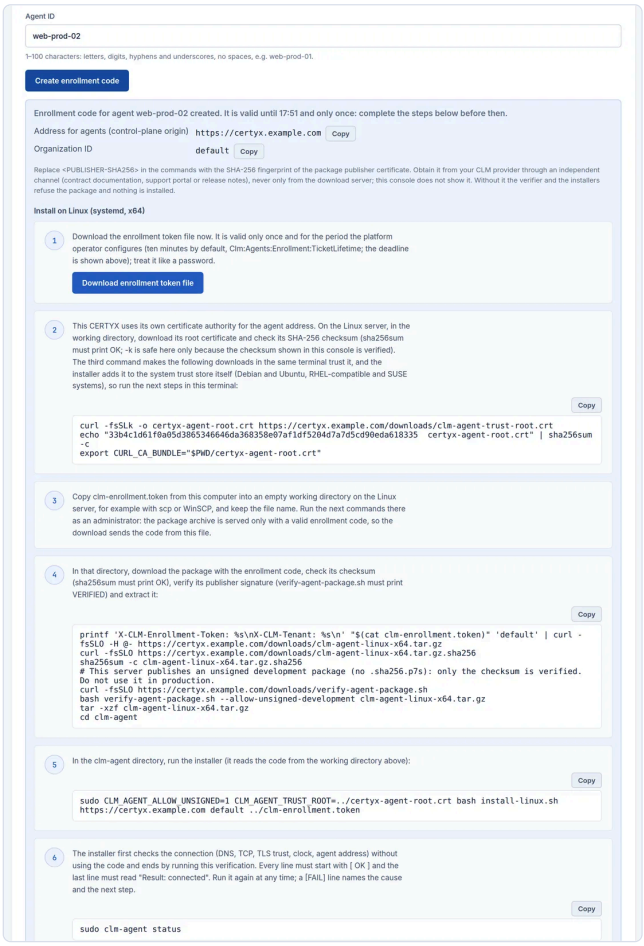
Every other request receives HTTP 401, which curl reports as `The requested URL returned error: 401` and PowerShell as `(401) Unauthorized` . It means that clm-enrollment.token is missing from the current directory, the code has expired or was already used, or the organization ID in the command is wrong: create a new code for the same agent ID, copy the new file into the working directory and run the commands again. Checksums (`.sha256`), signatures (`.p7s` , `.tsr`), release manifests (`.release.json` , `.sig`), the verifier scripts and `clm-agent-trust-root.crt` contain no code and stay public, so you can verify a package without a credential.

7 Linux with systemd

The quickest and safest way is the console: after Create enrollment code, Agents and destinations shows these steps with your real address and organization ID already filled in, ready to copy. Use the commands below when you cannot copy them from the console; then replace every placeholder in angle brackets, including the brackets, with the real value:

- `<CONTROL-PLANE-ORIGIN>` : the HTTPS address agents use, for example `https://agent.your-company.com` (no path, no trailing slash). The console shows it after Create enrollment code as "Address for agents": the agent host name when your operator configured one (`Clm:Ingress:AgentHosts`), otherwise the HTTPS address you use for the console (through a proxy on the agent port, for example `https://certyx.your-company.com:8443`). `https://clm.example.com` is only an example; the installer refuses it.
- `<ORGANIZATION-ID>` : your organization ID, shown in the same place as "Organization ID".
- `<PUBLISHER-SHA256>` : the SHA-256 fingerprint of the package publisher certificate. Obtain it from your CLM provider through an independent channel (contract documentation, support portal or release notes), never only from the download server, and keep it pinned (see "Package verification"). The console does not show it.
- `<ADMIN-USER>` and `<SERVER>` : your login on the Linux server and the server's host name, used only to copy the token file in step 2.

7.1 Step 1: create the enrollment code



Agents and destinations after “Create enrollment code”: address for agents, organization ID and the numbered steps with ready-to-copy commands (sample data)

In CERTYX → Agents and destinations, enter an agent ID (letters, digits, - and _; one ID per agent) and select Create enrollment code, then Download enrollment token file. Only administrators can create codes; an agent ID that is already registered is refused, so choose a new ID for a replacement agent. Creating another code for the same not-yet-enrolled agent ID is allowed; each code works on its own. The file clm-enrollment.token is valid for ten minutes and only once: complete steps 2–4 (copy, download and install) within ten minutes, or create a new code for the same agent ID. Treat it like a password and never paste its content into a command.

7.2 Step 2: copy the token file into a working directory

On the Linux server, create an empty working directory and copy clm-enrollment.token from the computer where you downloaded it into that directory, for example with `scp clm-enrollment.token <ADMIN-USER>@<SERVER>:` (your home directory on the server) or with WinSCP. Keep the file name. Run the commands of steps 3 and 4 from there.

7.3 Step 3: download, verify and extract the package

When the console shows the step "This CERTYX uses its own certificate authority" (the operator set `Clm:Ingress:AgentTrustRootFile`, for example the local test server), run it first, in the working directory and in the same terminal as the following steps, because the package download already needs this trust. The checksum is shown in the console next to the command; `-k` is safe here only because the downloaded file is compared with it, and `sha256sum` must print `certyx-agent-root.crt: OK`. The third command makes the following `curl` downloads of this terminal trust the root; the installer then adds it to the system trust store itself (`CLM_AGENT_TRUST_ROOT` in the install command, see step 4), so no trust-store command is typed by hand:

```
curl -fsSLk -o certyx-agent-root.crt <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-trust-root.crt
echo "<SHA256-FROM-CONSOLE>  certyx-agent-root.crt" | sha256sum -c
export CURL_CA_BUNDLE="$PWD/certyx-agent-root.crt"
```

Then, on the Linux server (x64 with systemd, outbound HTTPS to CERTYX), as an administrator in the working directory that holds clm-enrollment.token. The first command sends the code from that file, because the archive is served only with a valid enrollment code (see "Package downloads"):

```
printf 'X-CLM-Enrollment-Token: %s\nX-CLM-Tenant: %s\n' "$(cat clm-enrollment.token)" '<ORGANIZATION-ID>' | curl -fsSLO -H @- <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-linux-x64.tar.gz
```

```
curl -fsSLO <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-linux-x64.tar.gz.sha256
sha256sum -c clm-agent-linux-x64.tar.gz.sha256
curl -fsSLO <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-linux-x64.tar.gz.sha256.p7s
curl -fsSLO <CONTROL-PLANE-ORIGIN>/downloads/verify-agent-package.sh
bash verify-agent-package.sh --publisher-sha256 <PUBLISHER-SHA256> clm-agent-linux-x64.tar.gz
tar -xzf clm-agent-linux-x64.tar.gz
cd clm-agent
```

`sha256sum` must print `clm-agent-linux-x64.tar.gz: OK`; otherwise do not install. The checksum detects a damaged download. `verify-agent-package.sh` must print `VERIFIED`; `REFUSED: <reason>` means do not install (see "Package verification"). When your operator publishes a release signing key, also verify the signed manifest as described in step 2 of "Five steps". If `curl` reports a certificate problem, read "Private CA" below first; `The requested URL returned error: 401` on the first command means the code is missing, expired or used, or the organization ID is wrong (see "Package downloads").

7.4 Step 4: run the installer

In the clm-agent directory created by `tar` (the token file stays in the working directory above it, hence `../clm-enrollment.token`):

```
sudo CLM_AGENT_PUBLISHER_SHA256=<PUBLISHER-SHA256> bash install-linux.sh <CONTROL-PLANE-ORIGIN> <ORGANIZATION-ID> ../clm-enrollment.token
```

The fingerprint after `CLM_AGENT_PUBLISHER_SHA256=` is required: when the command copied from the console does not contain it, add it as shown. Before changing the host, the installer copies the package to a root-owned private staging directory under /var/tmp, verifies that copy (`SHA256SUMS`, its signature and the pinned publisher) and installs exactly the verified copy. `CLM_AGENT_PUBLISHER_CA_FILE` additionally requires the CA chain; `CLM_AGENT_VERIFIER=/path/verify-agent-package.sh` uses your separately downloaded verifier. `bash install-linux.sh --verify-only` checks the extracted package without installing. Without a pin, or with an unsigned or modified package, nothing is installed and the enrollment code is not used. `CLM_AGENT_ALLOW_UNSIGNED=1` is a development-only override that prints a warning. `CLM_AGENT_ENDPOINT_VERIFIER=<file>` additionally makes the host the independent endpoint verifier of a deployment target (see "Independent private-network verification").

With a private root (the console's trust step) the command shown by the console also passes `CLM_AGENT_TRUST_ROOT=../certyx-agent-root.crt`: the installer checks that the file is a CA certificate valid for at least another day and adds it under the name `certyx-agent-root.crt` to the system trust store of the distribution (Debian and Ubuntu: /usr/local/share/ca-certificates with `update-ca-certificates`; RHEL-compatible: /etc/pki/ca-trust/source/anchors with `update-ca-trust`; SUSE: /etc/pki/trust/anchors) before its connection check. An identical anchor is left as it is; when the connection check fails, an anchor added by this run is removed again, so the host trust store stays unchanged.

The installer checks every argument before it changes anything and names a wrong one together with the next step. It then runs `Clm.Agent check` against the address (DNS, TCP, TLS trust with online revocation, clock, agent routes); a failed step is printed with its cause and nothing is changed, and the enrollment code is not used, so you can fix the problem and run the same command again. It then creates the unprivileged clm-agent account, installs the release under /opt/clm-agent/releases/<version> with /opt/clm-agent/current pointing at it, enrolls (the agent generates its RSA 3072 key on this server; the key never leaves it), deletes its own copy of the token, keeps the private configuration, identity and state under /etc/clm-agent (`agent.json`, `identity.pfx`, `identity.password`, `state/`; owner-only for the clm-agent account) and writes and starts the systemd unit `/etc/systemd/system/clm-discovery.service`, which runs `/opt/clm-agent/current/Clm.Agent scan /etc/clm-agent/agent.json` as clm-agent. Finally it installs the helper `/usr/local/bin/clm-agent` (the package's `uninstall-linux.sh` stays in /opt/clm-agent/current for the removal), waits until CERTYX records the first contact of the running service, prints `sudo clm-agent status` and ends with "registered and running". Then delete clm-enrollment.token from the working directory; the code is used up. No raw-socket privilege or root daemon is required; root is needed only to create the account and the service.

7.5 Step 5 (optional): signed updates

Only when your CERTYX operator publishes a release signing key. Get release-signing.pub and its SHA-256 fingerprint from the operator through a trusted channel (not only from the same server), copy the key next to the installer and add it as a fourth argument:

```
sudo CLM_AGENT_PUBLISHER_SHA256=<PUBLISHER-SHA256> bash install-linux.sh <CONTROL-PLANE-ORIGIN> <ORGANIZATION-ID> ../clm-enrollment.token ./release-signing.pub
```

The installer prints the key fingerprint; compare it with the published one. The update service downloads new archives with this agent's client certificate, which the installer configures (`AgentConfigurationFile` in update.json). Without the fourth argument the agent works but is not updated automatically; to enable signed updates later, run `sudo bash upgrade-linux.sh ./release-signing.pub` from a newer extracted package (see "In-place upgrade").

7.6 Step 6: verify that the agent works

```
sudo clm-agent status
```

The installer already printed this once; run it at any time. Every line must start with `[ OK ]` and the last line must read `Result: connected`, for example:

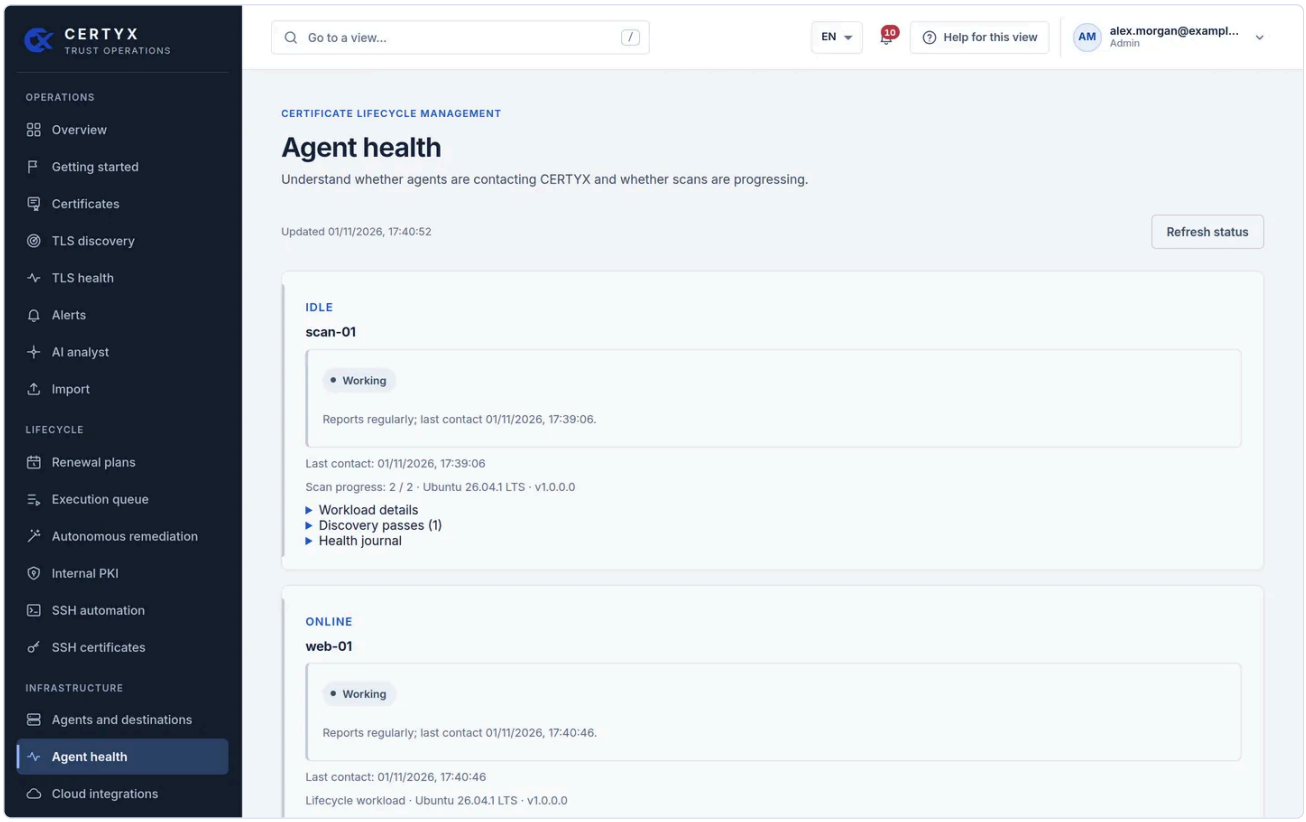
```
[ OK ] Service: clm-discovery.service is active (running).
CERTYX agent 1.0.0: status of /etc/clm-agent/agent.json
[ OK ] Configuration: agent address https://agent.your-company.com, organization acme.
[ OK ] Identity: SHA-256 3BCC36CD…43A67FC0, valid until 2027-04-08; the private key stays on this host.
[ OK ] DNS: agent.your-company.com resolves to 10.20.0.15.
[ OK ] TCP: connected to 10.20.0.15:443 in 5 ms.
[ OK ] TLS: certificate for agent.your-company.com issued by Company Issuing CA is trusted, matches agent.your-company.com, is not revoked and is valid until 2027-11-11.
[ OK ] Clock: this host's clock matches the server (difference 0 s).
[ OK ] Agent address: agent.your-company.com serves agents.
[ OK ] Registration: CERTYX accepts this host as agent "berlin-dc1" in organization "acme"; first contact 2026-10-10 18:30:33 UTC, last contact 2026-10-10 18:31:03 UTC.
Result: connected. CERTYX accepts this agent; its contact times are shown above and in CERTYX -> Agents and destinations.
```

It runs as the agent account and checks the service, the configuration, the local identity (certificate fingerprint and validity), name resolution, the TCP connection, TLS trust including revocation, the clock difference and whether CERTYX accepts the identity, with the first and last contact CERTYX recorded. Exit code 0 means working, 1 a problem (the first `[FAIL]` line names it, see "Troubleshooting"). `sudo clm-agent check` runs only the service and connection steps; `sudo clm-agent log [lines]` shows the last service log lines (default 50); `clm-agent version` prints the installed release. In CERTYX the console shows "Agent … works: first contact received at …" right after Create enrollment code once the service reports, and Registered agents shows the first and last contact ("no contact yet" while the service has not reached CERTYX). The agent then waits for its network profile ("No network profile is assigned") until you continue with step 4 of "Five steps". Stop scanning with `sudo systemctl stop clm-discovery.service` or by disabling the profile in CERTYX; `sudo systemctl status clm-discovery.service` and `sudo journalctl -u clm-discovery.service` show the service itself.

7.7 Private CA

When the CERTYX HTTPS certificate comes from your organization's own CA, the server must trust that root before the connection check of step 4. When the operator publishes the root with `Clm:Ingress:AgentTrustRootFile`, nothing is typed by hand: the console's trust step downloads it and checks its SHA-256 checksum, and the installer adds it to the system trust store (Debian and Ubuntu, RHEL-compatible and SUSE) with `CLM_AGENT_TRUST_ROOT` and removes it again after a failed check; `uninstall-linux.sh` removes it with the agent. Without a published root, get it from your PKI team and pass the file the same way: `sudo CLM_AGENT_PUBLISHER_SHA256=<PUBLISHER-SHA256> CLM_AGENT_TRUST_ROOT=<ROOT-CA-FILE> bash install-linux.sh …` (for the package download use `curl --cacert <ROOT-CA-FILE>`). The installer refuses a file that is not a CA certificate or expires within a day, and stops with a message on a distribution without `update-ca-certificates` or `update-ca-trust`. The agent also checks revocation of the CERTYX certificate online, so that certificate must name a CRL distribution point (or OCSP responder) the server can reach. Public CAs provide both.

7.8 Diagnosis in the console



Agent health: every agent with its state, last contact, workloads and the cause of a problem (sample data)

Agent health shows for every agent the exact cause CERTYX derives from its contact, identity and reports — not reporting since a time, no first contact yet, identity expired or expiring, clock skew (measured from the network map's time stamp), workload error with the fixed reason the agent reported, a workload that missed its window, missing configuration, reduced scan rate — with the facts and the remedies in order, each linked to where it is done. Every change of the cause is kept in the agent's health journal and the security journal (`agent.health.changed`). A lifecycle agent also reports its destinations (ID, DNS names, connector, installation mode, remote host of an SSH relay, deployment directory, files, check and reload commands, cloud resource, endpoint; no secrets, SSH identities or known_hosts content), which the renewal readiness of a certificate compares with the execution target: the target ID must equal a destination ID with the same names, purpose, installation mode and remote host, otherwise the agent refuses the job.

7.9 Troubleshooting on Linux

The installer, `Clm.Agent check <address>` and `sudo clm-agent status` print one line per step; fix the first `[FAIL]` line and run the command again. Their messages:

- `[FAIL] DNS: <name> cannot be resolved from this host (HostNotFound)`: the name is wrong or the DNS server of this host does not know it. Check the address from the console and `/etc/resolv.conf`; for a test, add the name to `/etc/hosts`.
- `[FAIL] TCP: cannot connect to <host> port <port> (… ConnectionRefused)`: nothing listens there. Check the port in the agent address; on the local test server start CERTYX with `CLM_AGENT_HOST` and allow inbound TCP 5443.
- `[FAIL] TCP: … (… TimedOut)`: a firewall drops the connection or there is no route. Allow outbound TCP to the port (and inbound on the server).
- `[FAIL] TLS: no TLS handshake with …`: the port does not speak HTTPS, for example the plain-HTTP console port 5180. Use the HTTPS agent address.
- `[FAIL] TLS: this host does not trust the issuer of the server certificate (…)`: add the private or local root (the console's trust step or "Private CA") and run the command again.
- `[FAIL] TLS: revocation of the server certificate cannot be checked: this host cannot download the revocation list http://…`: allow outbound HTTP to the named address (on the local test server: inbound TCP 5181 and a running `./start.sh`). `… the certificate names no CRL distribution point` means the operator must use a certificate from a CA that publishes a CRL.
- `[FAIL] TLS: a certificate of the chain is outside its validity period (…; this host's time is …)`: check this host's clock; if it is right, the operator must renew the server certificate.
- `[FAIL] TLS: the server certificate does not name <host>`: use exactly the address shown in the console; the certificate must contain that name or IP address.
- `[FAIL] Clock: this host's clock differs from the server by <n> minutes`: enrollment and authentication fail with more than a few minutes of difference. Run `sudo timedatectl set-ntp true` (or configure NTP) and try again.
- `[FAIL] Agent address: <address> does not serve agents (HTTP 404)`: you used the console address while the installation has a separate agent address; use the "Address for agents".

- `[FAIL] Registration: CERTYX does not accept this agent identity (HTTP 401)`: the agent was revoked, its certificate expired or it belongs to another organization or installation. Replace it as described for "already has an enrolled CERTYX agent".
- `[FAIL] Service: clm-discovery.service is inactive`: start it with `sudo systemctl start clm-discovery.service` and read `sudo clm-agent log`.
- `[FAIL] Configuration: cannot read the agent configuration`: run the command with sudo; if the file is missing the agent is not installed.

Download messages:

- `curl: (22) The requested URL returned error: 401` for `clm-agent-linux-x64.tar.gz`: the archive download did not carry a valid enrollment code. Run the command in the working directory that holds clm-enrollment.token; if it does, the code has expired or was used, or the organization ID is wrong. Create a new code, copy the new file and run the commands again (see "Package downloads").

Installer messages:

- `Package verification failed; nothing was installed.`: the package is unsigned, was changed after signing or was signed by another publisher than `CLM_AGENT_PUBLISHER_SHA256` names. Check the fingerprint (obtained from your CLM provider independently of the download), extract the package again from the verified archive and do not add or change files in it.
- `Package verifier missing; nothing was installed.`: `verify-agent-package.sh` is missing from the extracted package or `CLM_AGENT_VERIFIER` does not name a regular file. Extract the verified archive again or point `CLM_AGENT_VERIFIER` to the separately downloaded verifier.
- `Enrollment token file not found: <path>`: the third argument does not point to the token file on this server. Download it (step 1), copy it into the working directory (step 2), run the installer in the clm-agent directory below it and pass `../clm-enrollment.token`. The message also shows the current directory.
- `Argument 1 is still a placeholder`, `the example address from the guide` or `not a valid control-plane origin`: use the "Address for agents" from the console: `https://`, the host name and an optional port, without a path.
- `The enrollment code was rejected` (HTTP 400, 401, 403 or 409): the code expired (it is valid for the period set in `Clm:Agents:Enrollment:TicketLifetime`, ten minutes by default; the console shows the deadline), was already used, or the organization ID does not match. Create a new code for the same agent ID, copy the new file and run the same command again; the local key from the failed attempt is reused.
- `does not serve agents` (HTTP 404): you used the console address while your installation has a separate agent address. Use the "Address for agents" shown by the console.
- `cannot be reached` or `cannot be resolved`: check the address, port, DNS and outbound HTTPS firewall rule. `curl -s -o /dev/null -w '%{http_code}\n' <CONTROL-PLANE-ORIGIN>/health/live` must print 200.
- `does not trust the HTTPS certificate`: see "Private CA".
- `already has an enrolled CERTYX agent`: this server is enrolled already and nothing was changed; check it with `sudo clm-agent status`. To replace it, remove it with `sudo bash /opt/clm-agent/current/uninstall-linux.sh` (see "Removal on Linux"), revoke the agent in CERTYX, then create a code for a new agent ID (an agent ID can be enrolled only once) and install again.
- `clm-discovery.service is not running`: the installer prints the last log lines. Read more with `sudo journalctl -u clm-discovery.service -n 50 --no-pager`; failures are logged by category (exception type, HTTP status), never with tokens or keys.
- `This host does not run systemd`: use "Manual / container execution".

7.10 Removal on Linux

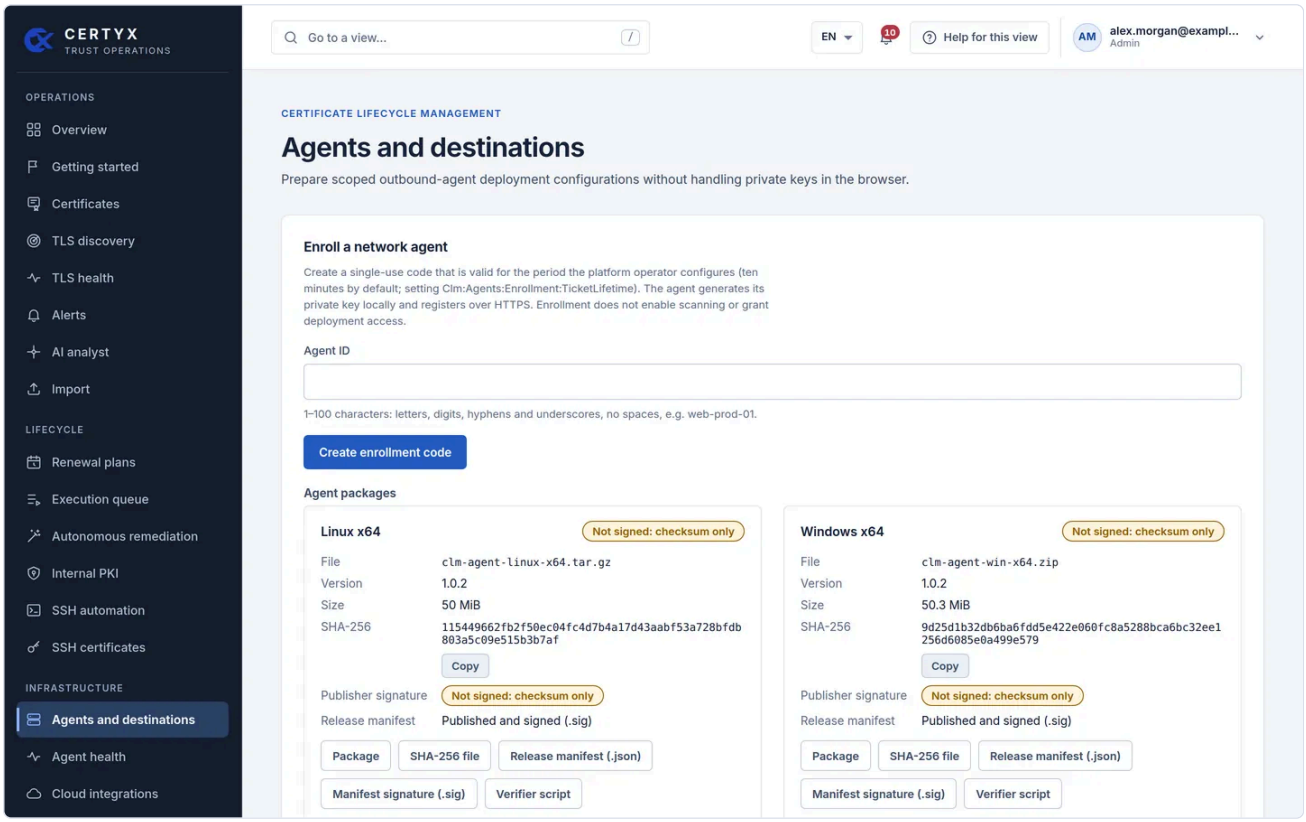
```
sudo bash /opt/clm-agent/current/uninstall-linux.sh
```

The script continues from a temporary copy (it deletes its own directory), stops, disables and deletes every unit the CERTYX scripts installed (`clm-discovery.service`, `clm-ssh.service` with its host-deployment drop-in, `clm-ssh-renewal.service` and `.timer`, `clm-endpoint-verifier.service`, `clm-agent-update.service` and `.timer`), removes the CERTYX block that `trust-user-ca` added to /etc/ssh/sshd_config (checked with `sshd -t`; a backup is kept and restored when sshd refuses the result; `--keep-ssh-trust` leaves sshd unchanged), the sudoers rule of the SSH host deployment and the `certyx-agent-root.crt` trust anchor the installer added (`--keep-trust-root` keeps it), and deletes /opt/clm-agent, /etc/clm-agent, /var/lib/clm-agent, /etc/clm-agent-update, /etc/clm-ssh, /etc/clm-verifier, /usr/local/bin/clm-agent and the clm-agent account. The private keys are deleted with these directories. Running it again on a clean host changes nothing. Then revoke the agent in CERTYX → Agents and destinations (Registered agents → Revoke agent): CERTYX stops accepting its identity only then.

8 Windows with Task Scheduler

As on Linux, the console shows these commands with your real values after Create enrollment code. The placeholders `<CONTROL-PLANE-ORIGIN>`, `<ORGANIZATION-ID>` and `<PUBLISHER-SHA256>` mean the same as in the Linux section; replace them including the angle brackets.

8.1 Step 1: create the enrollment code



Agents and destinations: enrollment, registered agents, network profiles and destination drafts (sample data)

As in Linux step 1: create the code in CERTYX and download clm-enrollment.token. It is valid only once and for the configured lifetime (`Clm:Agents:Enrollment:TicketLifetime`, ten minutes by default; the console shows the deadline); complete steps 2–4 within that time.

8.2 Step 2: copy the token file into a working folder

Create an empty working folder on the Windows server and copy clm-enrollment.token into it. Keep the file name. Run the commands of steps 3 and 4 in an administrator PowerShell from there.

8.3 Step 3: download, verify and extract the package

When the console shows the trust step for its own certificate authority, run it first in an administrator PowerShell; the second command must print `True`:

```
curl.exe -fsSLk -o certyx-agent-root.crt <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-trust-root.crt
(Get-FileHash certyx-agent-root.crt -Algorithm SHA256).Hash -eq '<SHA256-FROM-CONSOLE>'
Import-Certificate -FilePath .\certyx-agent-root.crt -CertStoreLocation Cert:\LocalMachine\Root
```

Then, in an administrator PowerShell on the Windows server (x64), in the working folder that holds clm-enrollment.token. The first command reads the code from that file into the `$clmDownload` headers that the archive download sends (see "Package downloads"):

```
$clmDownload = @{ 'X-CLM-Enrollment-Token' = (Get-Content .\clm-enrollment.token -Raw).Trim(); 'X-CLM-Tenant' = '<ORGANIZATION-ID>' }
Invoke-WebRequest <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-win-x64.zip -Headers $clmDownload -OutFile clm-agent-win-x64.zip -UseBasicParsing
Invoke-WebRequest <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-win-x64.zip.sha256 -OutFile clm-agent-win-x64.zip.sha256 -UseBasicParsing
(Get-FileHash clm-agent-win-x64.zip -Algorithm SHA256).Hash -eq (Get-Content clm-agent-win-x64.zip.sha256).Split(' ')[0]
Invoke-WebRequest <CONTROL-PLANE-ORIGIN>/downloads/clm-agent-win-x64.zip.sha256.p7s -OutFile clm-agent-win-x64.zip.sha256.p7s -UseBasicParsing
Invoke-WebRequest <CONTROL-PLANE-ORIGIN>/downloads/verify-agent-package.ps1 -OutFile verify-agent-package.ps1 -UseBasicParsing
Get-AuthenticodeSignature .\verify-agent-package.ps1
.\verify-agent-package.ps1 -PublisherSha256 <PUBLISHER-SHA256> -Archive .\clm-agent-win-x64.zip
Expand-Archive clm-agent-win-x64.zip -DestinationPath clm-agent
cd clm-agent
```

The `Get-FileHash` comparison must print `True` and `verify-agent-package.ps1` must print `VERIFIED`; otherwise do not install (see "Package verification"). `(401) Unauthorized` on the archive download means the code is missing, expired or used, or the organization ID is wrong (see "Package downloads"). You can also verify the signed manifest with PowerShell 7 or OpenSSL as described in "Signed updates".

8.4 Step 4: run the installer and check the task

In the clm-agent folder created by `Expand-Archive` (the token file stays in the working folder above it, hence `..\clm-enrollment.token`):

```
powershell -NoProfile -ExecutionPolicy Bypass -File .\install-windows.ps1 -ControlPlane <CONTROL-PLANE-ORIGIN> -Tenant <ORGANIZATION-ID> -TokenFile
..\clm-enrollment.token -PublisherSha256 <PUBLISHER-SHA256>
Get-ScheduledTask -TaskName 'CLM Discovery Agent' | Get-ScheduledTaskInfo
```

`-ExecutionPolicy Bypass` applies only to this one installer process. The installer checks the arguments first and names a wrong one with the next step (the same messages as on Linux), then runs `Clm.Agent.exe check` against the address without using the code. At the end it waits for the first contact and prints the status check. Run it again at any time in an administrator PowerShell; every line must start with `[ OK ]` and the last one must read `Result: connected`:

```
& "$env:ProgramData\CLM-Agent\Clm.Agent.exe" status "$env:ProgramData\CLM-Agent\agent.json"
```

The installer restricts ProgramData\CLM-Agent permissions, makes Administrators the owner, copies the token inside for enrollment and deletes that copy afterwards, and registers the scheduled task `CLM Discovery Agent`, which starts at system startup as LOCAL SERVICE, runs `C:\ProgramData\CLM-Agent\Clm.Agent.exe scan C:\ProgramData\CLM-Agent\agent.json` without a time limit and is restarted up to three times at one-minute intervals after a failure. The agent and its configuration (`agent.json`) live in `C:\ProgramData\CLM-Agent`, the identity (`identity.pfx`, `identity.password`) in `C:\ProgramData\CLM-Agent\identity` and the state in `state\`; only these two subdirectories are writable by LOCAL SERVICE: identity rotation stages the next identity (`identity.pfx.pending`) in the identity directory and renames it into place, and nothing else of the program directory can be changed by the task account. The installer refuses a non-empty ProgramData\CLM-Agent, copies the package into the restricted directory and verifies that protected copy (signed manifest, pinned publisher and the Authenticode signatures) before enrollment; a refused package is removed again and nothing is enrolled. `-PublisherSha256` is required; `-PublisherCaFile` additionally requires the CA chain, `-VerifierPath` uses a separately downloaded verifier and `-VerifyOnly` only checks the extracted package. `-AllowUnsignedDevelopment` is a development-only override that prints a warning. It does not create a Windows Service. Delete clm-enrollment.token from the working folder afterwards. Later versions are installed in place with `upgrade-windows.ps1` (see "In-place upgrade").

Automatic signed updates (optional): when your operator publishes a release signing key, add `-ReleaseKeyFile .\release-signing.pub` (obtained with its fingerprint through a trusted channel) to the install command. After enrollment the installer runs the installed `upgrade-windows.ps1 -EnableUpdates`, which pins the key and the publisher fingerprint in `C:\ProgramData\CLM-Agent-Update` (Administrators and LocalSystem only) and registers the scheduled task `CERTYX Agent Update`: daily at 03:00 with a random delay of up to six hours, as LocalSystem, it runs `upgrade-windows.ps1 -Scheduled`, the same signed `-Download` upgrade described in "In-place upgrade" (rollback included); a release that is not newer ends with exit code 0, and the output is appended to `C:\ProgramData\CLM-Agent-Update\update.log`. Check it with `Get-ScheduledTask -TaskName 'CERTYX Agent Update' | Get-ScheduledTaskInfo`. An installation without the key gains the task with its first signed upgrade (`-ReleaseKeyFile`) or with `upgrade-windows.ps1 -EnableUpdates -PublisherSha256 <PUBLISHER-SHA256> -ReleaseKeyFile .\release-signing.pub`.

The task has no console window: the agent writes its output to `C:\ProgramData\CLM-Agent\state\discovery.log` (time-stamped UTC lines, at most 5 MB; the previous part is kept as `discovery.log.1`). Read it in an administrator PowerShell with `Get-Content -Tail 50 C:\ProgramData\CLM-Agent\state\discovery.log`, the Windows counterpart of `sudo clm-agent log`. The SSH workload task writes `C:\ProgramData\CLM-SSH\state\ssh-workload.log`, whose last lines `configure-ssh-windows.ps1 status` shows. The log contains the agent's safe summaries only, never tokens, passwords or keys.

When CERTYX uses a private CA without a published root, import the root your PKI team gives you first: `Import-Certificate -FilePath <ROOT-CA-FILE> -CertStoreLocation Cert:\LocalMachine\Root`, where `<ROOT-CA-FILE>` is the path of the root certificate your PKI team gives you. When the console shows the trust step, use its commands instead. The CRL distribution point requirement from "Private CA" applies too.

Troubleshooting: the messages and remedies are those listed for Linux. To replace an installation, remove it with `uninstall-windows.ps1` (see "Removal"), revoke the agent in CERTYX and install again with a code for a new agent ID.

8.5 Removal

In an administrator PowerShell:

```
powershell -NoProfile -ExecutionPolicy Bypass -File "$env:ProgramData\CLM-Agent\uninstall-windows.ps1"
```

The script stops and removes the scheduled tasks `CLM Discovery Agent`, `CERTYX SSH Workload`, `CERTYX SSH Renewal`, `CERTYX Endpoint Verifier` and `CERTYX Agent Update`, removes the CERTYX block that `trust-user-ca` added to `C:\ProgramData\ssh\sshd_config` together with `certyx_user_ca.pub` (checked with `sshd -t`; a backup is kept and restored when sshd does not accept the result; `-KeepSshTrust` leaves sshd unchanged) and deletes `C:\ProgramData\CLM-Agent` (program, configuration, identity, endpoint verifier, state and logs), `C:\ProgramData\CLM-SSH` (SSH CA, keys and workload state) and `C:\ProgramData\CLM-Agent-Update` (the pinned release key, the publisher fingerprint and the update log). The private keys are deleted with these folders. Then revoke the agent in CERTYX → Agents and destinations (Registered agents → Revoke agent): CERTYX stops accepting its identity only then. An agent ID can be enrolled once; a new installation needs a code for a new agent ID.

9 Manual / container execution

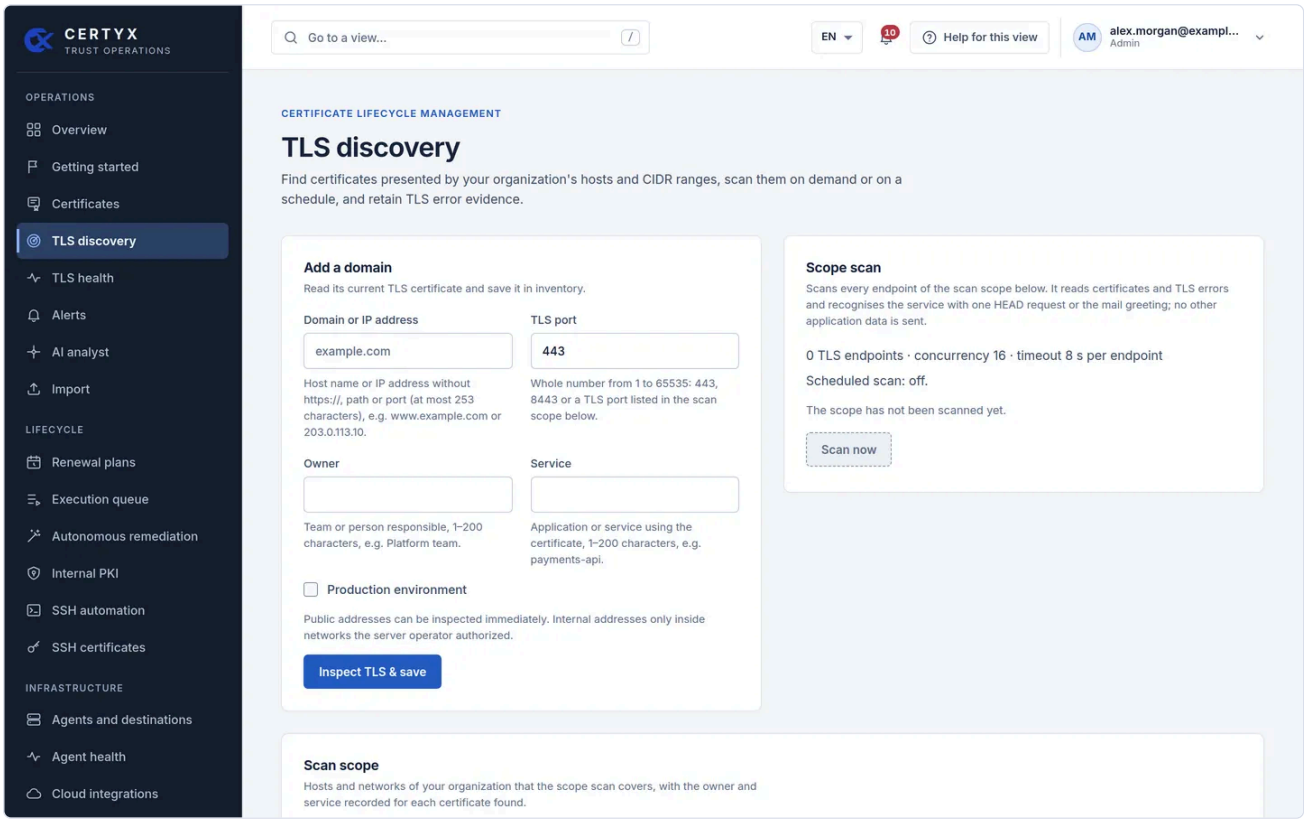
The same executable can run without an OS installer:

```
Clm.Agent enroll <CONTROL-PLANE-ORIGIN> <ORGANIZATION-ID> <ENROLLMENT-TOKEN-FILE> <CONFIG-FILE>
Clm.Agent scan <CONFIG-FILE>
Clm.Agent scan <CONFIG-FILE> --once
```

`<CONTROL-PLANE-ORIGIN>` and `<ORGANIZATION-ID>` are the values the console shows after Create enrollment code; `<ENROLLMENT-TOKEN-FILE>` is the downloaded clm-enrollment.token (owner-only, mode 600); `<CONFIG-FILE>` is the path where enrollment writes the agent configuration, for example /var/lib/clm-agent/agent.json; its directory also receives the identity files.

Mount configuration and identity files privately, persist the state directory, and provide the same routed network reachability as a host agent. Container bridge networks are not automatically the customer network: the network map of a container shows its bridge network, so approve the reachable CIDRs explicitly, and set `RestrictToAllowedCidrs` with `AllowedCidrs` when the container host must enforce its own boundary. --once processes one bounded batch (at most the profile's attempts per cycle), not the entire network.

10 Coverage and progress



TLS discovery: scope, progress and the endpoints and certificates found in the private network (sample data)

Each pass has up to three phases, always in this documented order:

1. **Liveness:** every address of the approved CIDRs (minus exclusions and, up to /30, the network and broadcast addresses) in ascending order. The agent tries the liveness ports one after another; an address is live when a port accepts or actively refuses (TCP reset) the connection. Addresses in the host's ARP neighbour table count as live without a connection.
2. **Common ports:** each live host in ascending order, its common ports in profile order, with a TCP connection and a direct TLS handshake, followed by the service identification described under "For the customer's security team".
3. **Full** (mode `full` only): every remaining port 1–65535 in ascending order, each port on all live hosts before the next port, so consecutive connections go to different hosts.

There is no randomization and no jitter. The agent never opens more than the profile's connections per host (1–4) to one host and starts connections at a steady rate of at most `ratePerSecond` (1–2000); the local `ProbeSpacingMilliseconds` can only make it slower. Outside the scan windows (weekly windows in the profile time zone; an end before the start continues on the next day, `24:00` ends at midnight, windows follow the local wall clock across daylight-saving changes) the agent opens no connections and reports the state `waiting-window` with the next window start. When more than `BackOffFailurePercent` (default 50 %) of the connection attempts of a common or full segment time out, the rate of the next segment is halved (never below `BackOffMinimumPercent`, default 10 %, of the approved rate) and raised again by `BackOffRestorePercent` (default 25 %) of the approved rate after each calmer segment; telemetry shows `backOff`. Telemetry also reports the phase, the position within the phase, live hosts and `baselineCompletedAt`, the time the first complete pass of the current scope finished.

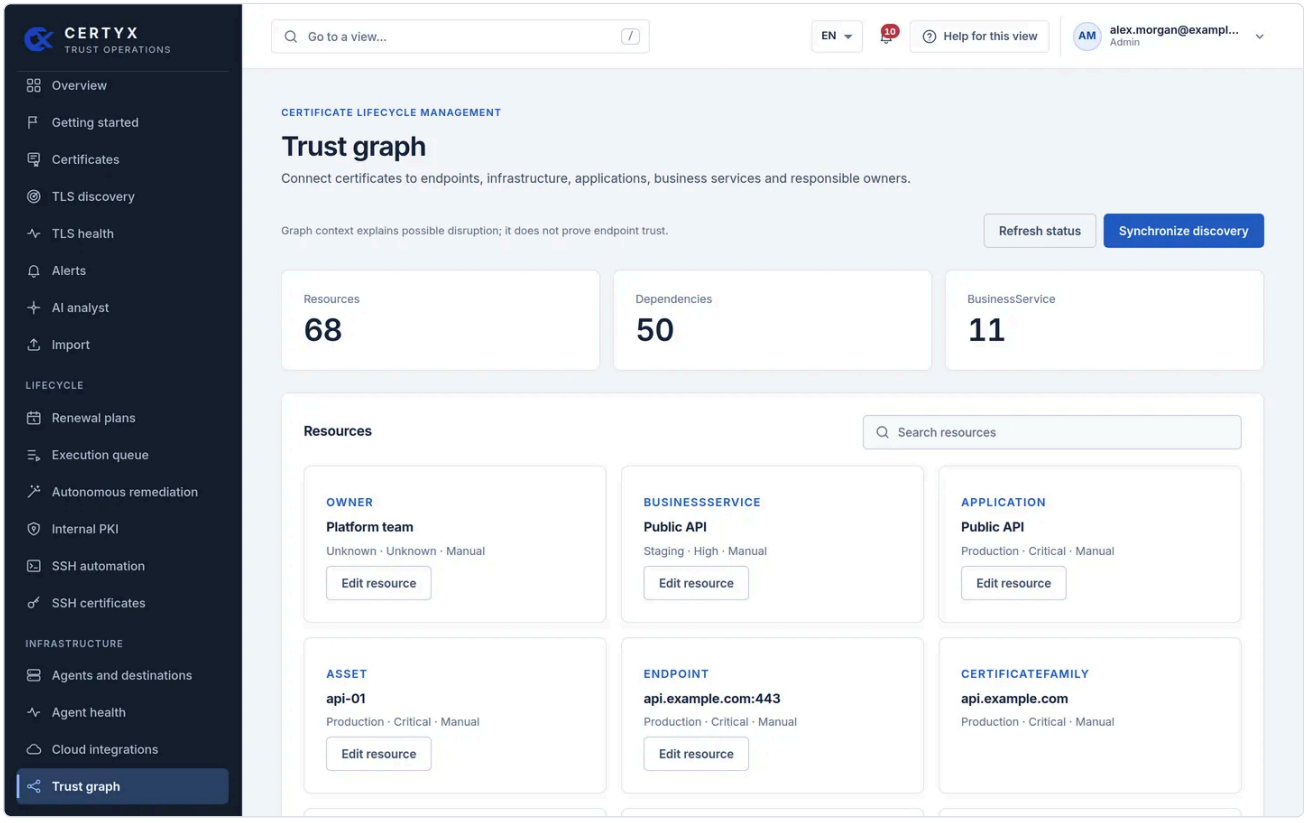
Progress is saved after every segment (`SegmentSize`, default 200 work items) in the owner-only checkpoint: phase, position and live hosts. The checkpoint is bound to a hash of the scope and ports only, so changing the rate, connections per host, windows or batch values continues the pass, while a changed scope, mode or port list restarts it. The profile is re-checked between segments; a batch ends after the profile's attempts per cycle and the next batch starts after the profile's interval. An observation refused by CLM with HTTP 400 (for example because the scope shrank meanwhile) is dropped, reported in telemetry, and the scan continues. A malformed profile is refused at validation and reported as invalid; the agent keeps running and retries after `RetrySeconds`.

Realistic durations are work items divided by the effective rate, which is the lower of `ratePerSecond` and `Concurrency` divided by the connect timeout when most attempts time out. Batches (`attemptsPerCycle`) only bound the work between profile refreshes and checkpoints and follow each other immediately; `intervalSeconds` is the pause after a complete pass:

- Liveness of a /24 with 4 liveness ports: at most about 1,000 attempts, about 10 seconds at 100 per second. A sparsely used /16: about 262,000 attempts, about 45 minutes at 100 per second or 9 minutes at 500 per second. Absent addresses time out, so 32 parallel connections with the 500 ms connect timeout allow only about 64 attempts per second (about 70 minutes).
- Common ports: live hosts × common ports, for example 300 hosts × 20 ports = 6,000 attempts, about one minute at 100 per second.
- Full mode: live hosts × 65,535 ports, for example 50 hosts = about 3.3 million attempts, about 4.6 hours at 200 per second or 55 minutes at 1,000 per second. Firewalls that drop packets turn most attempts into time-outs and back-off, which can stretch this to a day or more. With 10,000 attempts per cycle, 200 per second and a 300-second interval the average is about 29 attempts per second, so the same pass takes about 32 hours; shorten the interval when the windows and your security team allow it.

Absence of a discovered certificate is not proof of absence. Discovery currently detects direct TLS over IPv4 TCP. UDP/DTLS/QUIC, SMTP/LDAP STARTTLS, IPv6 address sweeps, SNI virtual-host enumeration, filesystem certificate stores and cloud connectors are not included in this scanner.

11 Network map



Trust graph: hosts, services and certificates with their dependencies (sample data)

At start and then every `NetworkMapIntervalMinutes` (default 60) the agent sends CLM the private IPv4 networks of its host: directly connected networks of active interfaces and destinations of the routing table (Linux `/proc/net/route`, Windows IP Helper routing table), plus private addresses of the ARP neighbour table (Linux `/proc/net/arp`, Windows IP Helper neighbour table) as liveness hints. Only RFC 1918 ranges and 100.64.0.0/10 are reported; default routes, loopback, link-local, multicast, public destinations and anything wider than /8 are dropped, and at most 1,024 networks and 4,096 neighbours are sent. Reading these tables sends no packets. The map is only a suggestion for the administrator: nothing is scanned until it is approved in a profile. If CLM refuses the map (for example an older server without this endpoint), the agent logs the HTTP status and tries again at the next interval; scanning is not affected. IPv6 neighbour tables are not read; on other operating systems only interface networks are reported.

12 For the customer's security team

Discovery is meant to be seen and recognised, not hidden. Agree the following with your security team before you enable a profile:

- **Source addresses:** the IP addresses of the hosts on which a discovery agent runs. Each agent connects only from its own host; CLM itself never connects into your network.
- **Targets and ports:** only the approved CIDRs minus exclusions; the liveness ports, then the common ports of live hosts and, in full mode, every TCP port 1–65535 of live hosts. These are ordinary TCP connections (complete handshake, no raw packets, no spoofed addresses) followed by a TLS handshake; the connection is closed once the certificate has been read and the service identified.
- **Service identification:** the TLS handshake offers the ALPN protocols `h2` and `http/1.1` (not on mail, LDAP, AMQP, MQTT and DNS-over-TLS ports) and records the TLS version, cipher suite, ALPN protocol and the issuer certificates the endpoint presents. Then, on an endpoint that negotiated an HTTP protocol or none on an HTTP port (443, 4443, 8443, 9443, 10443, 9000), the agent sends exactly one request — `HEAD / HTTP/1.1` with `Host` set to the address, `User-Agent: CERTYX-Discovery`, `Accept: */*` and `Connection: close` — and reads only the status line and headers (at most 16 KiB, 2 seconds); it never follows a redirect, never reads a body and never sends a `GET`, also not after `405`. An endpoint that negotiated HTTP/2 gets this request over one more connection that offers only HTTP/1.1. On SMTPS 465, IMAPS 993 and POP3S 995 it sends nothing and reads the first greeting line. Other non-HTTP ports (25, 110, 143, 587, 636, 853, 2376, 3269, 5671, 5986, 6443, 8883) receive no request. Web-server and IDS logs show these requests with the user agent `CERTYX-Discovery`. `"Settings": { "ServiceIdentification": { "IdentifyServices": false } }` in the scan configuration switches the request and the greeting read off; the server operator does the same for server-side scans with `Clm:Discovery:ServiceIdentification:IdentifyServices`.
- **Order and timing:** ascending addresses in a fixed order, no randomization and no jitter; only inside the approved windows in the profile time zone; at most `ratePerSecond` new connections per second per agent and at most 1–4 simultaneous connections per target host; automatic slow-down when many connections time out.
- **Register the agent as an approved scanner:** add the agent host addresses to the scanner or vulnerability-assessment exceptions of your IDS/IPS, firewalls, NAC and EDR, limited to the approved CIDRs and windows, so alerts are classified as expected activity instead of being suppressed globally. Without this registration full mode in particular looks like a port scan and may be blocked or reported as an attack.
- **Stopping:** disabling the profile in CLM stops scanning at the next segment; stopping the service (`systemctl stop clm-discovery.service` or the scheduled task) stops it at once; `RestrictToAllowedCidrs` lets the host owner limit the agent to locally approved networks whatever the profile says.
- **Records:** the agent log on the host shows every segment; CLM shows the agent telemetry (phase, progress, live hosts, `waiting-window`, back-off).

13 Operational settings

Each agent configuration file may contain an optional top-level `Settings` object. Omitted values keep the defaults below, which are the values the agent has always used. Values are checked when the command starts; an out-of-range value stops the command before any network call and is reported by exception type only (`InvalidDataException`). `enroll` does not write a `Settings` object; an administrator adds one to the protected configuration when needed. Settings never contain credentials; keep secrets in the protected files and references described in this guide.

- `scan` configuration: `RequestTimeoutSeconds` 20 (1–300), `IdentityRenewalDays` 30 (1–179), `ConnectTimeoutMilliseconds` 500 (1–60000), `HandshakeTimeoutSeconds` 3 (1–300), `ProbeSpacingMilliseconds` 0 (0–10000, local minimum spacing of connection starts; 0 leaves the pacing to the profile rate, a higher value only slows the agent down), `SegmentSize` 200 (1–10000, work items between profile checks, uploads, telemetry and checkpoints), `IdleIntervalSeconds` 30 (1–86400, pause when no profile is assigned and longest sleep while waiting for a scan window), `RetrySeconds` 30 (1–86400, pause after a failed cycle), `NetworkMapIntervalMinutes` 60 (1–1440, network map interval), `RestrictToAllowedCidrs` false (`true`: every profile CIDR must lie inside the local `AllowedCidrs`), `BackOffFailurePercent` 50 (1–100), `BackOffRestorePercent` 25 (1–100), `BackOffMinimumPercent` 10 (1–100), and the object `ServiceIdentification` with `IdentifyServices` true (`false` stops the HEAD request and greeting read), `TimeoutMilliseconds` 2000 (100–60000), `MaximumResponseHeaderKilobytes` 16 (1–64), the port lists `HttpPorts`, `HttpRequestExcludedPorts`, `AlpnExcludedPorts` and `GreetingPorts` (at most 256 ports each; defaults in DISCOVERY-FULL.md section 8) and `UserAgent` `CERTYX-Discovery`. Mode, ports, rate, connections per host, windows, attempts per cycle and the batch interval come from the approved profile.
- `ssh manage` and `ssh monitor-krl` configuration: `RequestTimeoutSeconds` 20 and `IdentityRenewalDays` 30, at the top level of the file (not inside `Identity`). Their polling intervals stay `PollSeconds` and `Subscription.PollSeconds`.
- Lifecycle agent configuration: `PollSeconds` 10 (1–86400), `RequestTimeoutSeconds` 15 (1–300), `OperationTimeoutSeconds` 600 (1–86400, upper bound for one job; a cloud certificate store target adds three times its convergence window, for AWS ACM 18 minutes), `HeartbeatSeconds` 25 (1–119, lease renewal; must stay below the 120-second server lease and the operation timeout), `TrustChainTimeoutSeconds` 3 (1–300).
- Connector limits, in every configuration file including the local `ssh` commands (`renew`, `publish-krl`, `sync-krl`, `apply-krl` and the authority commands, which read `Settings` from the first configuration file): `Settings.Connectors` may contain `Process`, `Nginx`, `Deployment`, `HostDeployment`, `Acme`, `Dns`, `Vault`, `Secrets` and `CloudVerification` (lifecycle agent and `secret-check`; `HostDeployment` holds the limits of the file-based connectors and the SSH relay: `CommandTimeout` 00:01:00 per check, reload, key generation or SSH round trip, `VerificationTimeout` 00:00:30, `RetryDelay` 00:00:00.5, `RevocationTimeout` 00:00:03, `MaximumReadBytes` 1048576 (4096–16777216) and `ChownExecutable` `/usr/bin/chown`; `CloudVerification` applies to AWS ACM targets — when you set it, keep `ConvergenceWindow` at least `00:06:00`, the AWS default when the object is omitted) and `SshKeygen`, `SshHostService`, `SshHostDeployment` and `SshKrlSubscription` (SSH commands). Durations use `hh:mm:ss` text, must be positive and at most one hour; `SshHostService` second counts are 1–300. CONFIGURATION.md lists every value and default under "Customer-side connector limits". An omitted object keeps the connector defaults.

Example: `"Settings": { "SegmentSize": 100, "RetrySeconds": 60 }` or `"Settings": { "Connectors": { "Nginx": { "VerificationTimeout": "00:00:30" } } }`. Protocol and security bounds are fixed: 5 observations per upload, the discovery scope limits described in step 4, response size limits, 1 MiB journal entries and RSA-3072 identities valid for 180 days.

14 Revocation and recovery

Use Revoke agent in CLM to reject subsequent authenticated calls. Disabling a profile stops subsequent batches; an in-progress probe may finish. Identity lifetime is 180 days; discovery agents automatically rotate local keys and identities 30 days before expiry (default of `IdentityRenewalDays`). To rotate at once, for example after a suspected key exposure, run `Clm.Agent rotate-identity <config-file>` as administrator (Linux: `sudo /opt/clm-agent/current/Clm.Agent rotate-identity /etc/clm-agent/agent.json`; Windows: `& "$env:ProgramData\CLM-Agent\Clm.Agent.exe" rotate-identity "$env:ProgramData\CLM-Agent\agent.json"`); the running agent rotates at its next identity check and `status` then shows the new fingerprint. Preserve pending files during recovery. A 24-hour overlap handles lost acknowledgments; see AGENT-IDENTITY-ROTATION.md. The existing lifecycle agent remains available separately and requires reviewed CA/deployment configuration. Network enrollment does not activate its deployment target. Keep the local identity files if enrollment acknowledgments are uncertain; do not replace a key blindly. Failed setup is reported by exception type plus, for HTTP failures, the status code or transport category (for example `HttpRequestException (HTTP 401)` or `HttpRequestException (SecureConnectionError)`), without printing tokens; a refused enrollment reports the enrollment's own status rather than that of the recovery probe.

15 Package production

Operators build the bundles with `bash scripts/package-agent.sh linux-x64` (or linux-arm64, win-x64, win-arm64). Publish each architecture separately. Runtime packages may require access to Microsoft's NuGet feed. Set `CLM_AGENT_VERSION` (major.minor.patch) and `CLM_RELEASE_SIGNING_KEY` (path to the offline ECDSA P-384 release key) to produce the signed release manifest for updates. With the production backends `pkcs11` and `external` the script refuses to run without an ECDSA P-384 `CLM_RELEASE_SIGNING_KEY` and checks the signature it wrote; only the development backends `pfx` and `unsigned-development` may produce an unsigned manifest, marked "DEVELOPMENT PACKAGE", which agents refuse as an update (and installers refuse the unsigned publisher signature of `unsigned-development` unless their development override is set). Keep both signing keys off the CLM servers. Publisher code signing additionally needs OpenSSL 3 and, for Windows packages, `osslsigncode` (Authenticode on Linux). The script refuses to run until a signing backend is selected with `CLM_SIGN_BACKEND`; `scripts/sign-agent-package.sh` performs the signing:

- `pkcs11` (production): the key stays in an HSM, smart card or cloud HSM with a PKCS#11 module. Set `CLM_SIGN_PKCS11_MODULE` (vendor module), `CLM_SIGN_KEY_URI` (`pkcs11:` URI with `pin-source=file:` pointing to an owner-only PIN file; PIN values in the URI are refused) and `CLM_SIGN_CERT_CHAIN` (publisher certificate first, then intermediates). The OpenSSL PKCS#11 engine (libp11) is required.
- `external` (production): `CLM_SIGN_AUTHENTICODE_COMMAND` (called as `command <in> <out>`) and `CLM_SIGN_CMS_COMMAND` (called as `command <in> <out.p7s>`, DER detached CMS with SHA-256) wrap operator-managed tools, for example signtool on a Windows build host or a tool bound to a cloud key service. Both must use the same publisher certificate.
- `pfx` (development only): `CLM_SIGN_PFX_FILE` and `CLM_SIGN_PFX_PASSWORD_FILE`, owner-only and outside the repository, plus `CLM_SIGN_ALLOW_DEVELOPMENT_KEY=1`. The publisher certificate must be a code-signing leaf (`CA:FALSE`): Windows refuses Authenticode signatures by a CA certificate even when it is trusted, so `sign-agent-package.sh` stops for Windows packages signed with one. `scripts/create-development-publisher.sh <new-directory>` creates a development publisher CA and leaf with the PFX and password file; on a Windows development host trust its `publisher-ca.crt` (`Import-Certificate -FilePath publisher-ca.crt -CertStoreLocation Cert:\LocalMachine\Root`) or pass it as `-PublisherCaFile`, never on customer hosts.
- `unsigned-development`: hash manifests without signatures; verifiers and installers refuse these packages unless their development override is set.

A Windows build also writes the Authenticode-signed `verify-agent-package.ps1` next to the archives; publish it with them, so `Get-AuthenticodeSignature .\verify-agent-package.ps1` (the console's command) shows the pinned publisher.

`CLM_SIGN_TIMESTAMP_URL` adds RFC 3161 time stamps to Authenticode and CMS signatures so packages stay verifiable after the publisher certificate expires. `CLM_SIGN_PUBLISHER_SHA256` re-verifies the output against the expected fingerprint before the build succeeds. The release step prints the publisher fingerprint; publish it through your independent customer channel, never only next to the download. Signing keys, PINs and PFX files are never written into the package or the build output and must never be committed. Publish the archive together with its `.sha256`, `.sha256.p7s`, (when used) `.p7s.tsr`, `.release.json` and `.release.json.sig` files. When container images are built without the signed archives (deploy/), copy the complete package set into the directory configured as `Clm:AgentPackages:Directory` (the deployment mounts `$CERTYX_ROOT/agent-packages`); it is served at `/downloads/` before wwwroot/downloads, on the console host and on the agent host. The verifiers are served separately at `/downloads/verify-agent-package.sh`, `/downloads/verify-agent-package.ps1` and `/downloads/verify-release.sh`.

15.1 Reproducible builds, SBOM and provenance

`scripts/build-agent.sh <rid> <directory>` builds the unsigned agent binaries reproducibly: in the .NET SDK image pinned by digest in `deploy/Dockerfile`, from `git archive` of the commit, with deterministic compiler output (`ContinuousIntegrationBuild`, mapped source paths, the commit as source revision) and `SOURCE_DATE_EPOCH` set to the commit time. It writes `clm-agent-<rid>.build.tar.gz` (sorted, owner 0, fixed times), `clm-agent-<rid>.cdx.json` (CycloneDX 1.5 SBOM of every NuGet package and runtime pack, from the restore result, without network tools) and `clm-agent-<rid>.build.json` (version, commit, SDK, image and the SHA-256 of every binary). Two builds of the same commit give identical files. `scripts/package-agent.sh` uses the same script (`--local`, with the SDK on the build machine) and places the SBOM and the build record into every package, where `SHA256SUMS` covers them.

The GitHub workflow `agent-release.yml` (tag `agent-v<version>`) builds all four runtimes this way, attests the build provenance (SLSA, `actions/attest-build-provenance`, when `CERTYX_ATTESTATIONS=true`) and, after approval in the `agent-release` environment, attaches the builds to a draft release. The signing keys are not in GitHub. Verify before signing:

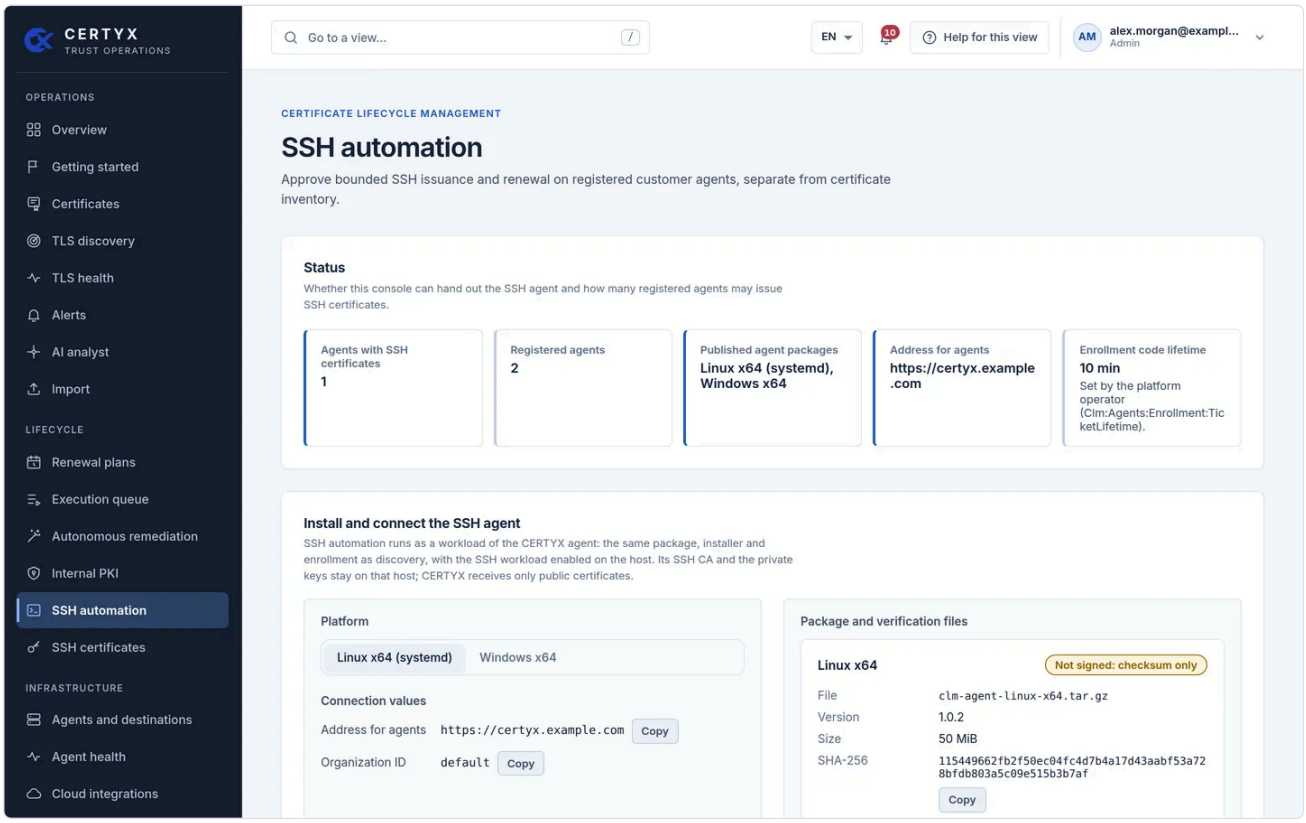
1. Provenance: `gh attestation verify clm-agent-linux-x64.build.tar.gz --repo <owner>/<repository>` (also for the `.cdx.json` and `.build.json`).
2. Reproducibility, independently of GitHub: `git checkout agent-v<version>` and `CLM_AGENT_VERSION=<version> bash scripts/build-agent.sh linux-x64 /tmp/rebuild`; `cmp` the `.build.tar.gz` with the downloaded one (or compare the `files` hashes of the two `.build.json`).
3. Package and sign offline from exactly these binaries: `CLM_AGENT_BUILD=<directory with the three files> CLM_AGENT_VERSION=<version> CLM_SIGN_BACKEND=pkcs11 CLM_RELEASE_SIGNING_KEY=<key> bash scripts/package-agent.sh linux-x64`; the script refuses binaries that differ from their build record.

On a customer host, the package's `clm-agent-<rid>.build.json` names the commit and the SHA-256 of the binary (`sha256sum Clm.Agent`); Windows binaries carry the Authenticode signature added after the build, so their hash differs from the build record; check Windows builds through the `.build.tar.gz` of the release (step 2).

References: https://learn.microsoft.com/en-us/dotnet/core/deploying/ and https://learn.microsoft.com/en-us/powershell/module/scheduledtasks/new-scheduledtaskprincipal

16 SSH

16.1 SSH workload: download, install and configure (Linux and Windows)



SSH automation: install and connect the SSH agent, managed profiles and approvals (sample data)

The SSH workload is part of this agent; there is no separate SSH package. In CERTYX → **SSH automation** → **Install and connect the SSH agent** choose Linux x64 or Windows x64: the panel shows the package card of that platform (version, size, SHA-256 with a Copy button, signature state and download buttons for the package, its `.sha256`, `.sha256.p7s` when signed and the verifier; the package button works in your signed-in console session), creates the enrollment code and shows the commands of this guide filled in (address for agents, organization ID): copy clm-enrollment.token into an empty working directory on the server first, because the package download there sends the code (see "Package downloads"). After installation:

1. Grant **SSH certificates** to the agent in that panel (or Agents and destinations → Enable SSH operations); grant **SSH installation** only when the agent should install host or user certificates.
2. Linux: `sudo bash /opt/clm-agent/current/configure-ssh-linux.sh init` — Windows (administrator PowerShell): `powershell -NoProfile -ExecutionPolicy Bypass -File "$env:ProgramData\CLM-Agent\configure-ssh-windows.ps1" init`. It creates the host's SSH CA (`/etc/clm-ssh` owned by clm-agent, mode 0700; `C:\ProgramData\CLM-SSH`, SYSTEM and Administrators only) and prints `CA fingerprint: SHA256:…`.
3. Create the profile with that fingerprint in SSH automation, let a second administrator approve it and download it (**Download profile**, `ssh-profile-<ID>.json`).

4. … `configure-ssh-linux.sh add-profile ./ssh-profile-<ID>.json` or … `configure-ssh-windows.ps1 add-profile .\ssh-profile-<ID>.json`: writes `managed.json`, runs one verified `ssh manage --once` cycle and starts `clm-ssh.service` (Linux, user clm-agent) or the scheduled task **CERTYX SSH Workload** (Windows, LocalSystem). A failed cycle prints its cause, for example `HttpRequestException (HTTP 403)` when the SSH capability is missing; nothing is started.
5. Optional `trust-user-ca`: the local sshd trusts user certificates of this CA (`TrustedUserCAKeys`, checked with `sshd -T`, backup restored when sshd does not reload). On Windows Server 2025 the default sshd_config admits only the groups `administrators` and `OpenSSH Users` (`AllowGroups`): add accounts that log in with user certificates to `OpenSSH Users` (`Add-LocalGroupMember -Group 'OpenSSH Users' -Member <ACCOUNT>`), and start the OpenSSH server once before `trust-user-ca`, because it creates `sshd_config` at its first start.
6. Verify: `sudo clm-agent status` (Windows: `Clm.Agent.exe status agent.json`) prints `[ OK ] Capabilities: discovery, ssh; …`; `configure-ssh-linux.sh status` / `configure-ssh-windows.ps1 status` show the service or task; the SSH automation table shows *SSH certificates: Granted* and the SSH workload report.

Discovery and the SSH workload share the enrolled identity; rotation is serialized by `identity-rotation.lock`. For verified Linux host-certificate installation (**SSH installation** granted), `sudo bash /opt/clm-agent/current/configure-ssh-linux.sh enable-host-deploy` lets clm-agent run exactly `sshd -t` and the sshd reload through sudo (visudo-checked rule and service drop-in; `disable-host-deploy` removes them). Details, Windows ACL rules and host-installation targets: SSH-LIFECYCLE.md.

SSH local operations are available on Linux and on Windows (OpenSSH for Windows) with OpenSSH ssh-keygen installed. They issue Ed25519 user/host certificates and generate a KRL under an explicitly approved local CA/principal policy. See SSH-LIFECYCLE.md and the SSH workspace guide. Platform-approved SSH jobs and verified Unix host rotation are available through separately authorized managed configuration. Issuance alone does not install a user certificate or trust policy.

Unattended local SSH renewal without reporting to CERTYX (`Clm.Agent ssh renew`) is installed with `sudo bash /opt/clm-agent/current/configure-ssh-linux.sh enable-renewal ./ssh-profile-<PROFILE-ID>.json` (Windows: `configure-ssh-windows.ps1 enable-renewal .\ssh-profile-<PROFILE-ID>.json`). The script writes `renewal.json` from the local CA policy and the approved profile, runs one renewal cycle and enables `clm-ssh-renewal.timer` (a one-cycle service a minute under `clm-agent`, without network access, journal identifier `clm-ssh-renewal`; Windows: scheduled task `CERTYX SSH Renewal` as LocalSystem, log `state\ssh-renewal.log`). A cycle in which a profile is failed, blocked or in policy conflict ends with exit code 3 and leaves `clm-ssh-renewal.service` failed (`systemctl --failed`); `renewal-status` shows the last cycle, `disable-renewal` removes the timer, the service and `renewal.json` (certificates and keys stay). A profile is renewed either by `add-profile` (platform-approved jobs that report to CERTYX) or by `enable-renewal`, never by both. See `SSH-LIFECYCLE.md` for profiles, recovery and statuses.

17 Customer-controlled secrets and KRL subscriptions

See CUSTOMER-VAULTS.en.md for Azure/AWS/GCP and HashiCorp Vault references and secret-check preflight. See SSH-LIFECYCLE.md for publish-krl/sync-krl, signed one-hour freshness, separate relay permissions and recovery. The examples/ssh-krl-publisher.json and examples/ssh-krl-subscription.json templates require replacement of every placeholder and a nonzero organization UUID. Standalone sync-krl reports errors only locally; central per-consumer status comes from `ssh monitor-krl` with a separately enrolled identity that is explicitly granted KRL consumer reporting (shown in Agent health).

18 SSH workload user identities

Current Unix agents also support approved User profiles with installation enabled and matching local UserTargets. See SSH-LIFECYCLE.md and examples/ssh-user-target.json. The agent runs under the workload account, requires an existing working private bundle, verifies certificate authentication against separately pinned server trust and rolls back failed activation. Workloads must use current/identity and current/identity-cert.pub. Upgrade the agent before assigning user installation; Windows/human account provisioning and remote CA trust setup remain separate.

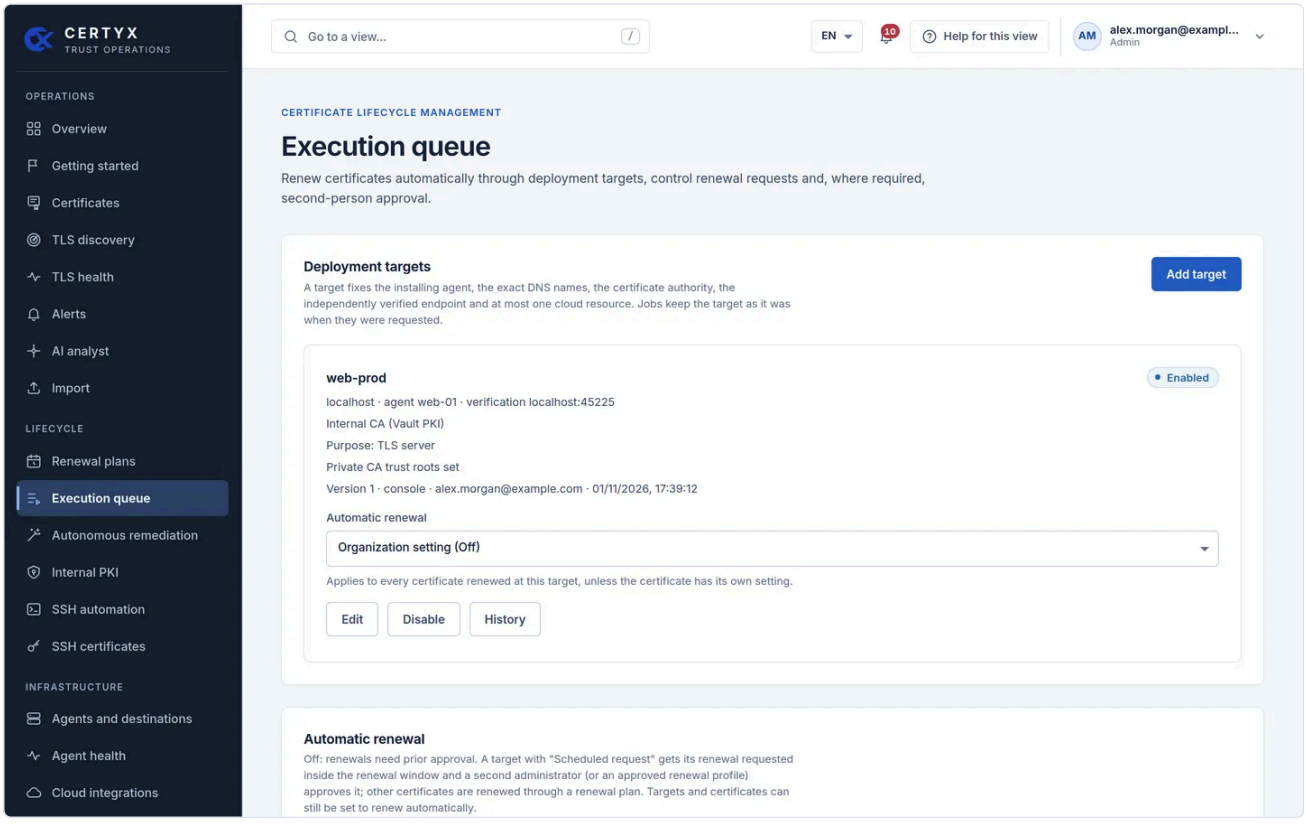
19 TLS client certificate destinations

A lifecycle destination can receive a TLS client certificate (users, devices, services in mutual TLS) instead of a server certificate. Set `Purpose` to `client` (or `server-client` for both usages; omitted means `server`) and list the identities in `Client`: `Emails` (rfc822Name, e.g. `jane.doe@example.com`), `UserPrincipalNames` (e.g. `jdoe@corp.example.com`), `Uris` (e.g. `spiffe://example.org/ns/prod/sa/payments`) and an optional `CommonName` (1–64 characters; required when only URIs are given). `Domains` may be empty (`[]`) for a client certificate. These values must equal the console target exactly, otherwise the agent refuses the job as outside its local scope. Client purposes require `AuthorityKind: "vault"` and a Vault role with `client_flag=true` (`server_flag=false` for client-only); a client destination with ACME or DigiCert, or with a cloud resource, is refused at start-up with that reason. A client-only destination has no endpoint: leave `VerifyHost` empty and `Executable` empty, or set `Executable` / `Arguments` to a fixed absolute command (for example a service restart) that runs after activation. The agent writes `current/fullchain.pem` and `current/privkey.pem` (owner-only) and, when `Pkcs12PasswordReference` names a protected customer secret, `current/bundle.p12` (PBES2 AES-256/SHA-256). It then verifies locally that the installed key matches, the certificate permits clientAuth and the chain builds to `TrustRootsFile`; a failure restores the previous generation. Unix only; client certificates are not installed into the Windows certificate store or over an SSH relay.

```
{
    "Id": "vpn-jane",
```

```
    "Domains": [],
    "Purpose": "client",
    "Client": { "Emails": ["jane.doe@example.com"], "UserPrincipalNames": ["jdoe@corp.example.com"] },
    "DeploymentDirectory": "/etc/clm/client/vpn-jane",
    "KeyDirectory": "/var/lib/clm-agent/keys",
    "AuthorityKind": "vault",
    "VaultOrigin": "https://vault.example.com",
    "VaultMount": "pki_int",
    "VaultRole": "client-users",
    "VaultTokenFile": "/etc/clm/secrets/vault-token",
    "TrustRootsFile": "/etc/clm/trust/internal-roots.pem",
    "Executable": "",
    "Arguments": [],
    "VerifyHost": "",
    "VerifyPort": 0,
    "LifetimeHours": 720,
    "Pkcs12PasswordReference": { "Provider": "protected-file", "Location": "/etc/clm/secrets/vpn-jane-p12.password" }
}
```

20 Deployment connectors and SSH relay



Execution queue: deployment targets and jobs that install renewed certificates and verify each installation (sample data)

A renewed TLS server certificate is installed in one of two modes:

- **Agent on the host** (`local-agent`): the lifecycle agent runs on the machine whose service uses the certificate, generates the key there and installs the files there.
- **SSH relay** (`ssh-relay`): a lifecycle agent installs over SSH on another host that has no agent. The key is generated on that host and never leaves it; the relay receives only the CSR.

Without an agent on the host or a relay with SSH access to it, CERTYX cannot install a certificate automatically; discovery, monitoring, renewal plans and reminders still work, and the certificate is then installed by hand.

20.1 Connectors

A destination names its connector in `Connector`. Without `Connector` (or with `nginx` and neither `Files` nor `Remote`) the original Nginx layout applies unchanged: generations in `DeploymentDirectory`, `current/fullchain.pem` and `current/privkey.pem`, and `Executable` / `Arguments` started with `-t` and `-s reload`. Every other connector publishes the files listed in `Files` at the paths the service already uses:

Connector	Required files	Default check	Default reload	Key file mode	Notes
`pem-files`	any of the file kinds	none	none	0600	Generic; set `CheckCommand` and `ReloadCommand` for the service.
`nginx`	`FullChain` + `PrivateKey`, or `Combined`	`/usr/sbin/nginx -t`	`/usr/sbin/nginx -s reload`	0600	`ssl_certificate` / `ssl_certificate_key`.
`apache`	`FullChain` + `PrivateKey`, or `Certificate` + `Chain` + `PrivateKey`	`/usr/sbin/apachectl -t`	`/usr/sbin/apachectl graceful`	0600	`SSLCertificateFile` (full chain since Apache 2.4.8), `SSLCertificateKeyFile`, `SSLCertificateChainFile` for the separate chain.
`haproxy`	`Combined` (full chain followed by the key)	`/usr/sbin/haproxy -c -f /etc/haproxy/haproxy.cfg`	`/usr/bin/systemctl reload haproxy`	0600	`bind :443 ssl crt <Combined>`.
`lighttpd`	`FullChain` + `PrivateKey`, or `Combined`	`/usr/sbin/lighttpd -tt -f /etc/lighttpd/lighttpd.conf`	`/usr/bin/systemctl reload lighttpd`	0600	`ssl.pemfile` and `ssl.privkey` (lighttpd 1.4.53 or later).
`caddy`	`FullChain` + `PrivateKey`; `Group` required	`/usr/bin/caddy validate --config /etc/caddy/Caddyfile`	`/usr/bin/caddy reload --config /etc/caddy/Caddyfile --force`	0640	`tls <FullChain> <PrivateKey>` in the site block; Caddy then does not obtain this certificate through its own ACME.
`traefik`	`FullChain` + `PrivateKey`	none	none	0600	File provider with `watch: true`; place the files in the directory the provider watches so the rename triggers the reload.
`envoy`	`FullChain` + `PrivateKey`; `Group` required	none	none	0640	File-based SDS (`path_config_source` with `watched_directory` on the certificate directory); the rename triggers the rotation. The bootstrap needs `node` with `id` and `cluster` for SDS.
`cribl`	`FullChain` + `PrivateKey`; `Group` required	none	`/usr/bin/systemctl restart cribl`	0640	Point the Cribl TLS settings (API server or a Source) at the files (`certPath`, `privKeyPath`); Cribl Edge: `ReloadCommand ["/usr/bin/systemctl","restart","cribl-edge"]`. Run one destination per Leader or Worker host. Without systemd use `["/opt/cribl/bin/cribl","restart"]`.
`tomcat`	`Pkcs12` or `JavaKeystore`; `Group` and `Pkcs12PasswordReference` required	none	`/usr/bin/systemctl restart tomcat`	0640	`certificateKeystoreFile` with `certificateKeystoreType="PKCS12"` and the same password; set `ReloadCommand` to your unit name (for example `tomcat10`).
`java-keystore`	`JavaKeystore`; `Pkcs12PasswordReference` required	none	none	0600	keytool imports the PKCS#12 bundle into `KeystoreType JKS` (default) or `PKCS12` with `KeystoreAlias`; set `ReloadCommand` to restart the application.
`postfix`	`FullChain` + `PrivateKey`	`/usr/sbin/postfix check`	`/usr/sbin/postfix reload`	0600	`smtpd_tls_chain_files = <PrivateKey>, <FullChain>`. Verification needs implicit TLS: `VerifyPort` 465.
`dovecot`	`FullChain` + `PrivateKey`	`/usr/bin/doveconf -n`	`/usr/bin/doveadm reload`	0600	`ssl_cert = <<FullChain>` and `ssl_key = <<PrivateKey>` (Dovecot 2.3; 2.4 uses `ssl_server_cert_file` and `ssl_server_key_file`). Verification needs implicit TLS: `VerifyPort` 993 or 995.
`kubernetes-secret`	none; `Kubernetes` object	none	none	—	`kubectl create secret tls … --dry-run=client` and `kubectl apply` on the deployment host (`tls.crt` = full chain, `tls.key`); `VerifyHost` is the Ingress or Gateway that serves it. Set `ReloadCommand` (for example a `kubectl rollout restart`) when the workload does not watch the secret.
`windows-store`	none; `WindowsStore` object	—	optional `ReloadCommand`	—	Windows agent only: LocalMachine store, non-exportable machine key.
`iis`	none; `WindowsStore` with the binding	—	`netsh http update sslcert`	—	Windows agent only; the HTTPS binding must exist (created once in IIS Manager).

Fields of a connector destination (all from the protected local configuration; a job only selects the destination):

- `Files`: `Certificate` (leaf), `Chain` (issuing chain without the leaf), `FullChain` (leaf and chain), `PrivateKey`, `Combined` (full chain followed by the key), `Pkcs12` (AES-256 PKCS#12) and `JavaKeystore`, each an absolute path of letters, digits and `. _ / @ + = , : % -`, in an existing directory that not everyone may write, without symbolic links in the path, distinct, and outside the generation store. `Owner` and `Group` set the owner and group of every published file (the agent or the relay account must be allowed to change ownership, normally root); `Mode` (default `0644`, no write access for group or others) applies to public files, `KeyMode` (default of the connector, `0600` or `0640`, never readable by others) to files that contain the key. `KeystoreType` (`JKS` or `PKCS12`), `KeystoreAlias` (default `certyx`) and `KeytoolExecutable` (default `/usr/bin/keytool`, agent on the host only) apply to `JavaKeystore`.
- `CheckCommand` and `ReloadCommand`: argument arrays that start with an absolute executable, for example `["/usr/sbin/nginx","-t"]`; arguments use the same character set (no shell, quotes, `$`, `;` or `|`). They replace the connector defaults; `[]` switches a default off. Over SSH they run behind `Remote.SudoPrefix`.
- `Pkcs12PasswordReference`: the customer secret reference of the PKCS#12 and keystore password (CUSTOMER-VAULTS.en.md); over SSH the password reaches the remote host on the standard input of the encrypted session, never on a command line.
- `Kubernetes`: `KubeconfigFile`, `Namespace`, `SecretName`, optional `Context`, and `KubectlExecutable` (default `/usr/local/bin/kubectl`, agent on the host only).
- `WindowsStore`: `StoreName` (`My` or `WebHosting`), `BindingHostName` (SNI binding) or `BindingAddress` (default `0.0.0.0`), `BindingPort` (443), `BindingAppId` (IIS: `{4dc3e181-e14b-4a21-b022-59fc669b0914}`) and `NetshExecutable`.
- `DeploymentDirectory`: on the agent's host, private (0700); it holds the deployment journal and lock and, on the agent's own host, the generation store. `VerifyHost` and `VerifyPort` name the TLS endpoint the agent checks after the reload (required).

What the agent does for a connector destination: it completes an interrupted deployment first; creates the store (0700) and an RSA-3072 key with its CSR on the deployment host (the CSR must carry a valid signature and exactly the approved DNS names); has the CA sign it; checks that the certificate carries exactly that public key and those names, permits TLS server authentication and chains to `TrustRootsFile`; stores a new generation (`privkey.pem` 0600, `cert.pem`, `chain.pem`, `fullchain.pem`, the bundles); publishes every file atomically (a new file next to the target, then a rename) with its owner, group and mode; runs the check and the reload; and verifies over TLS that `VerifyHost:VerifyPort` presents exactly the new certificate. CERTYX then verifies the endpoint independently, as for every server target. The store keeps the current and the previous generation. When the check, the reload or a verification fails, the previous generation is published again, checked, reloaded and verified, and the job reports `rolled-back`. Before the first deployment the agent copies the files that are published at that moment (byte for byte, with mode and owner) as the baseline it restores; when none of the files exists there is nothing to restore and the agent stops with "Intervention required" and keeps `journal.json` in the deployment directory: check the service, then remove the journal. A baseline that has only a `Combined`, PKCS#12 or keystore file is restored without a fingerprint check, because reading it would copy its key.

Examples: `examples/agent.deploy-<connector>.json` for every connector and `examples/agent.ssh-relay.json`.

20.2 SSH relay

The relay is an enrolled lifecycle agent with **Enable certificate lifecycle execution** (capability `deployment`). Every remote destination must also be listed in `RelayDestinations` of its configuration, and the deployment target must say installation mode **SSH relay** with the same remote host (Execution → Deployment targets → Installation); the relay refuses any job whose target names another mode or host. Requirements on the remote host: OpenSSH server, a POSIX login shell for the relay account (sh, bash, dash or zsh), the usual base tools (`mktemp`, `find`, `chmod`, `chown`, `mv`, `cp`, `sed`, `sort`), OpenSSL 1.1.1 or later in `/usr/bin` or `/usr/local/bin`, keytool for Java keystores, kubectl for Kubernetes, and `ss` (iproute2) for `relay-check`. Only TLS server certificates are relayed; client certificates and the Windows store need an agent on the host.

Steps 2, 4, 5 and 6 on the remote host are one command of the agent package: copy `configure-relay-host-linux.sh` (from `/opt/clm-agent/current` on the relay), the relay's public key and the helper (step 4) to the remote host and run, as root:

```
sudo sh configure-relay-host-linux.sh enable --account certyx-deploy --public-key ./id_ed25519.pub \
  --helper ./certyx-relay-helper --from 10.0.0.5 --root /etc/haproxy/certs --root /var/lib/certyx-relay \
  --allow "/usr/sbin/haproxy -c -f /etc/haproxy/haproxy.cfg" --allow "/usr/bin/systemctl reload haproxy"
```

It creates the account (key login only), writes `authorized_keys` with `restrict` and `from=`, installs the helper, writes the roots file, creates missing directories, writes the sudoers rule for the helper and exactly the `--allow` commands (checked with `visudo`) and prints the host key fingerprints to pin; `status --account certyx-deploy` shows the setup and `disable --account certyx-deploy` removes it (published certificates and store directories stay). It works with `useradd` (Debian, Ubuntu, RHEL, SUSE) and BusyBox `adduser` (Alpine). The steps below describe what it does and how to do it by hand.

Step by step (example: HAProxy on `web-01.internal.example`, relay account `certyx-deploy`):

1. **Relay key** (on the relay host, as root): `install -d -m 700 /etc/clm-agent/ssh` and `ssh-keygen -t ed25519 -N '' -C certyx-relay -f /etc/clm-agent/ssh/id_ed25519`. The private key stays owner-only on the relay; it is never sent to CERTYX.
2. **Remote account** (on the remote host): `useradd --system --create-home --shell /bin/sh certyx-deploy` and put the public key into `~certyx-deploy/.ssh/authorized_keys` (directory 700, file 600) with restrictions, for example `restrict,from="10.0.0.5" ssh-ed25519 AAAA… certyx-relay` (the relay's address).
3. **Host key pin**: on the remote host read the fingerprint `ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key.pub -E sha256`. On the relay run `ssh-keyscan -t ed25519 -p 22 web-01.internal.example > /etc/clm-agent/ssh/known_hosts`, `chmod 600 /etc/clm-agent/ssh/known_hosts` and compare `ssh-keygen -lf /etc/clm-agent/ssh/known_hosts -E sha256` with the fingerprint from the remote host. Put that `SHA256:…` value into `HostKeyFingerprints`. The agent refuses a known_hosts file that holds another key for the host, wildcard or negated entries, `@cert-authority` or `@revoked` lines; OpenSSH then enforces it with strict host key checking (no trust on first use). For a non-standard port the entry is `[host]:port`.

4. **Relay helper** (on the remote host): print it from the signed agent on the relay with `sudo /opt/clm-agent/current/Clm.Agent relay-helper > certyx-relay-helper`, copy it over and install it as root: `install -o root -g root -m 0755 certyx-relay-helper /usr/local/libexec/certyx-relay-helper`; compare `sha256sum` on both hosts. The helper performs only fixed operations (write, copy, read public files, create and remove store directories, key and CSR generation, PKCS#12, keystore, Kubernetes secret, listeners) and finds its tools in a fixed PATH. Without `HelperPath` the relay sends the same script through `/bin/sh -c`; that needs no installation but cannot be limited by sudoers.
5. **Allowed directories**: `/etc/certyx-relay/roots` (owner root, mode 0644, one absolute directory per line) lists where the helper may write and read, for example `/etc/haproxy/certs` and `/var/lib/certyx-relay`. It is required when the helper runs as root and also applies without sudo; every path must lie below one of them. List only the certificate directories and the store, never a web root or a directory other accounts can write. The helper reads back only public certificate files and its generation pointer, never a key, combined file or keystore.
6. **sudo** (when the account is not root; edit with `visudo -f /etc/sudoers.d/certyx-relay`): ``` Defaults:certyx-deploy !requiretty certyx-deploy ALL= (root) NOPASSWD: /usr/local/libexec/certyx-relay-helper, /usr/sbin/haproxy -c -f /etc/haproxy/haproxy.cfg, /usr/bin/systemctl reload haproxy ``` List exactly the helper and the check and reload commands of the destination, with their arguments. Set `"SudoPrefix":["/usr/bin/sudo","-n"]`. Do not allow `/bin/sh` or any editor: that grants full root.
7. **Destination** in the relay's lifecycle configuration (`examples/agent.ssh-relay.json`): ```json "RelayDestinations": ["web-01-haproxy"], "Destinations": [{ "Id": "web-01-haproxy", "Domains": ["www.example.com"], "Connector": "haproxy", "DeploymentDirectory": "/var/lib/clm-agent/relay/web-01-haproxy", "Files": { "Combined": "/etc/haproxy/certs/www.pem" }, "Remote": { "Host": "web-01.internal.example", "Port": 22, "User": "certyx-deploy", "KnownHostsFile": "/etc/clm-agent/ssh/known_hosts", "HostKeyFingerprints": ["SHA256:..."], "IdentityFile": "/etc/clm-agent/ssh/id_ed25519", "StoreDirectory": "/var/lib/certyx-relay/web-01-haproxy", "SudoPrefix": ["/usr/bin/sudo", "-n"], "HelperPath": "/usr/local/libexec/certyx-relay-helper" }, "VerifyHost": "www.example.com", "VerifyPort": 443, "AuthorityKind": "vault", "...": "CA fields as for every destination" }] ``` `Remote` fields: `Host` (DNS name or IP address), `Port` (22), `User`, `KnownHostsFile` and `IdentityFile` (absolute local paths, owner-only 0600), `HostKeyFingerprints` (1–8 pinned `SHA256:` values, for a key rotation list both), `StoreDirectory` (on the remote host, 0700: keys and generations), optional `JumpHost` (`Host`, `User`, `Port`, `HostKeyFingerprints`; reached with the same identity and known_hosts file), `ConnectTimeoutSeconds` (10, 1–120), `SshExecutable` (`/usr/bin/ssh`), `SudoPrefix` and `HelperPath`. `DeploymentDirectory` stays on the relay and holds only the journal and lock.
8. **Check without changing a certificate**: `sudo /opt/clm-agent/current/Clm.Agent relay-check <lifecycle-config.json> [destination-id]` checks the pin, the authentication, the helper version, the store directory, names the process listening on `VerifyPort` with the suggested connector (shown in the readiness of the certificate as "Service found by relay-check") and runs the configuration check. Exit code 0 means ready.
9. **Target**: in Execution → Deployment targets choose **Installation: Agent as SSH relay to a host without an agent** and the remote host (`Remote.Host`), with the relay agent as installing agent. The renewal readiness of the certificate then shows the relay, the remote host and the published files, and blocks with a clear reason when mode or host differ.

The relay connects with the system OpenSSH client and command-line options only (`-F /dev/null`): `BatchMode=yes`, `StrictHostKeyChecking=yes` against `KnownHostsFile`, public key authentication with `IdentityFile` only (`IdentitiesOnly=yes`, no ssh-agent), no agent or X11 forwarding, no port forwarding or tunnel, no local command, no connection sharing (`ControlMaster=no`) and `ConnectTimeout`. Every remote word is single-quoted, so paths and arguments stay literal; check and reload commands run with their standard streams on /dev/null, so a restart that starts a daemon (for example `cribl restart`) returns, and only the exit code counts. Each relay deployment is recorded in the security journal (`execution.relay.deployment.started|succeeded|failed|rolled-back` with agent, target, remote host and names). One relay should own a remote destination; two relays deploying to the same files are not coordinated.

21 Saved lifecycle result delivery

Persisted lifecycle results are retransmitted after temporary connection failures, request timeouts and HTTP 408/429/500/502/503/504, using PollSeconds and the OperationTimeoutSeconds bound. The same job, lease and result are reused; issuance and installation are never replayed. Restart recovery renews the lease while delivering the saved result. Authentication, authorization, expired/reconciled leases and TLS trust failures stop for reconciliation. A lost lease cannot be renewed by this mechanism; an outage exceeding the lease or operation limit still requires administrator reconciliation.

22 Protected configuration reads

All agent JSON configuration readers reject symbolic files or parent directories, Unix group/other permissions, files over 1 MiB and invalid UTF-8 before deserialization. Protect the configuration with chmod 600 on Unix. Relative command-line paths are resolved to an absolute path and receive the same checks. UTF-8 with or without BOM is supported. Settings and asynchronous readers use the same rules; no configuration values are included in failure output. On Windows the same checks read the ACL: a private file or directory must be owned by LocalSystem, Administrators or the agent's own identity, and no other principal may read or change it (deny entries are ignored). Files owned by an individual administrator account are refused; the installer sets Administrators as owner.

23 Independent private-network verification

For endpoints reachable only inside the customer network, use a separately enrolled endpoint verifier. In Agents and destinations, under Registered agents, select **Enable certificate lifecycle execution** for the installer and **Enable independent endpoint verification** for the observer (capabilities `deployment` and `endpoint-verification`). Use different agent identities and private keys. Discovery enrollment alone grants neither permission. Enroll both through the existing one-use-token HTTPS workflow; keep each generated identity and password file on its respective customer host.

An organization administrator sets the customer verifier agent ID on the deployment target in the Execution workspace (Deployment targets), alongside the verification host, port and health path; the API field is `verificationAgentId`. A change invalidates existing automatic-renewal scope approval. Leave the verifier ID absent to retain control-plane verification. Remote mode requires an enrolled installer: set both `tenant` and `clientCertificatePasswordFile` in its lifecycle configuration. The password file takes precedence over the legacy password environment variable. Enrolled lifecycle and verifier identities rotate automatically before expiry.

Install the verifier (one command). When the deployment target names the verifier agent, its card in Execution → Deployment targets shows **Set up the independent endpoint verifier**: select **Enable independent endpoint verification** for that agent, use **Download verifier scope** (`endpoint-verifier-<TARGET-ID>.json`: target ID, verification host, port, health path and the target's trust roots, public data only), copy the file to the verifier host and run, in the directory of the file:

```
sudo bash /opt/clm-agent/current/configure-verifier-linux.sh add-target ./endpoint-verifier-<TARGET-ID>.json
```

```
powershell -NoProfile -ExecutionPolicy Bypass -File "$env:ProgramData\CLM-Agent\configure-verifier-windows.ps1" add-target .\endpoint-verifier-<TARGET-ID>.json
```

The script writes the verifier configuration from the agent's own enrolled identity and the scope (Linux: `/etc/clm-verifier/verifier.json` and `roots/<TARGET-ID>.pem`, owner `clm-agent`, modes 600 and 700; Windows: `C:\ProgramData\CLM-Agent\verifier`, owned by Administrators and readable by LOCAL SERVICE), checks it with the verifier's own scope rules, runs one verified cycle (`verify-endpoints --once`) and starts the service: `clm-endpoint-verifier.service` under the `clm-agent` account (read-only system, `NoNewPrivileges`, restarted after a failure and together with `clm-discovery.service`, so a signed update also starts the verifier from the new release) or the scheduled task `CERTYX Endpoint Verifier` (LOCAL SERVICE, log `state\endpoint-verifier.log`). While CERTYX answers HTTP 403 because verification is not granted yet, the script waits up to ten minutes for the grant (`CLM_VERIFIER_GRANT_WAIT` or `-GrantWaitSeconds` in seconds; 0 fails at once and names the console step). Running it again with the same target replaces that target; another target file adds a target. `remove-target <TARGET-ID>` stops verifying one target (the service is removed with the last one), `disable` removes the service, the configuration and the trust roots, and `status` shows the service, its last log lines and the agent status; `uninstall-linux.sh` and `uninstall-windows.ps1` remove everything with the agent. A new agent receives the role during installation: add `CLM_AGENT_ENDPOINT_VERIFIER=./endpoint-verifier-<TARGET-ID>.json` to the Linux install command (Windows: `-EndpointVerifierTarget .\endpoint-verifier-<TARGET-ID>.json`); the installer then runs the same step after the agent is registered, waits for the grant and exits with code 3 when the agent runs but the verifier could not be enabled.

The verifier reaches CERTYX with the trust store of its host (the installer's trust step) and checks the endpoint against the trust roots of the scope; a scope without trust roots (a target whose roots are a server-side file) needs the issuing CA in the host trust store. Production requires HTTPS, trusted chains, server authentication, online revocation and an accurate clock. A configuration written by hand from `examples/endpoint-verifier.json` (`identity` is the observer's enrollment configuration) follows the same rules; `testMode` disables online revocation and is accepted only when the control plane and every target are loopback addresses; never enable it for customer workloads. No issuing credentials, installation paths or workload private keys belong in the verifier configuration.

After installation, the existing lease remains active while the installer retries only its persisted result. A separate observer retrieves a 60-second request bound to tenant, job revision, lease, target and expected SHA-256 fingerprint. It validates the live TLS chain, endpoint name, server EKU, revocation, exact certificate and configured HTTP health check, then returns public certificate evidence. CLM accepts success only after matching proof; a failed or absent observer cannot produce successful deployment. The observer cannot widen its locally approved target scope from a server request. Self-verification and shared current/pending identity keys are rejected.

Execution recovery uses a fresh observation request and retains independent approval and optimistic revision checks. The UI waits on pending observations for a bounded period. A timeout leaves the job for reconciliation; it never silently repeats issuance or installation. Agent health shows this workload separately, including safe failure and missed-report states. Healthy discovery cannot conceal a failed verifier. Consult the local service log for exception category and check scope, trust roots, network reachability, health URL, clock and capability assignment; private keys and CA credentials are never included in telemetry. Revoke or remove the verifier capability to stop further accepted observations.

23.1 Lifecycle installer diagnostics

Enrolled lifecycle agents report fixed, safe starting/completed/error reasons through their deployment capability. Agent health displays Deployment separately from discovery. During active operations health is refreshed with each lease heartbeat and advertises the configured heartbeat interval; idle cycles use the poll interval. Enrolled lifecycle polling must be at least five seconds. A missing report becomes an error after the reporting window, even when discovery remains healthy. Starting or a completed cycle is process health, not proof that a certificate was installed: consult the corresponding execution job and independent endpoint evidence. Unknown issuance/install side effects still stop for reconciliation. A reporting failure does not replay side effects; missing telemetry is detected by the server. Legacy static-registry installers are not registered fleet-health agents.

23.2 Managed deployment paths

Local key and PEM deployment directories must have link-free absolute paths, including their parents. CERTYX creates owner-only directories and refuses linked/replaced roots before writing keys or deployment state. Private keys and journals must remain owner-only (Linux 600); generated directories use 700. Existing shared journal or lock files require correcting permissions before restarting. Recovery refuses linked, shared or oversized

journals. The single relative `current` link is permitted only for a managed UUID generation; generation directories and certificate files must themselves be protected and link-free. No configuration should point at a user home, shared upload or externally writable deployment tree. These path checks do not replace host access controls.

23.3 AWS ACM

AWS imported-certificate target: see AWS-LIFECYCLE.md and examples/agent.aws-vault.json. Replace every identifier and domain, enroll the installer and separate verifier, grant deployment explicitly, and approve a server target with the same AwsAcm scope. Protect the existing certificate key/reference and prepare rollback with aws-adopt before enabling jobs. The example uses Vault issuance; it is not a template for a publicly trusted CA. AWS role credentials are resolved locally by the SDK.

24 Public CAs: DNS-01, EAB and DigiCert

An ACME destination can use `dns-01` validation (`AcmeChallengeType`, required for wildcard certificates) with a DNS provider in `AcmeDns` (Route 53, Azure DNS, Google Cloud DNS or Cloudflare, limited to one zone) and bind its account to a commercial CA contract (`AcmeEabKeyId` and `AcmeEabHmacReference`). A `digicert` destination orders or reissues certificates in DigiCert CertCentral within the purchase policy, and `Clm.Agent digicert-collect` writes a snapshot of purchased certificates to import in the console. API and HMAC keys are customer secrets resolved locally. Details: CA-CONNECTORS.md.

24.1 Discovery identity recovery

Discovery startup performs local validation/locking only. Rotation and provider calls run in the retryable cycle, so a transient startup outage does not exit the continuously running agent. A failed rotation acknowledgement marks the connection for rebuilding even when the next rotation reports no change; the old client is disposed only after a replacement is successfully constructed. Failed telemetry cannot mask the recoverable cycle failure. Request timeouts and HTTP 408/429/500/502/503/504 retry after Settings.RetrySeconds; authentication/authorization, TLS trust failures and user cancellation stop. `--once` still exits nonzero on a failed cycle. No trust or scope checks are bypassed. Discovery state/lock paths refuse links and require owner-only protection; existing broadly accessible state is rejected rather than chmodded into acceptance.

25 Signed updates

Releases are signed offline: `scripts/package-agent.sh` writes `clm-agent-<rid>.release.json` (version, runtime, archive name, size, SHA-256) and, with `CLM_RELEASE_SIGNING_KEY`, its detached ECDSA P-384/SHA-384 signature. Create the key once on an offline machine: `openssl ecparam -name secp384r1 -genkey -noout -out release-signing.key` and `openssl ec -in release-signing.key -pubout -out release-signing.pub`; publish the public key and its fingerprint (`openssl pkey -pubin -in release-signing.pub -outform DER | sha256sum`). CLM only serves these files from `/downloads/` and cannot forge an update.

When install-linux.sh receives the public key, it installs it as /etc/clm-agent-update/release-signing.pub (root-owned), writes /etc/clm-agent-update/update.json (including `"AgentConfigurationFile":"/etc/clm-agent/agent.json"`) and enables the daily `clm-agent-update.timer`. The root update service runs `Clm.Agent update /etc/clm-agent-update/update.json`, which:

- downloads the manifest and signature over HTTPS and verifies them with the pinned key; another key, an unsigned manifest or another runtime is refused;
- installs nothing when the published version is the same or older (no downgrade);
- downloads the archive with the enrolled agent's client certificate and organization (from `AgentConfigurationFile`), because the control plane serves archives only to an authorized request (see "Package downloads"), checks its size and SHA-256 against the signed manifest, and extracts it without links, absolute paths or `..` entries;
- runs the new binary with `--version`, which must report the signed version, then atomically repoints /opt/clm-agent/current, restarts clm-discovery.service and waits for `systemctl is-active`;
- when the restart or health check fails, points `current` back to the previous release and restarts it (exit code 3); the previous release is always kept.

update.json fields: `ReleaseOrigin` (HTTPS), `Rid`, `ReleaseKeyFile`, `InstallRoot`, `RestartCommand` and optional `HealthCommand` (absolute executables with fixed arguments), optional `TrustRootsFile` for a private CA, `AgentConfigurationFile` (the enrolled agent's `agent.json`, whose client certificate and organization authorize the archive download), `HealthTimeoutSeconds` (5–900, default 60) and `KeepReleases` (2–10). Exit codes: 0 installed or already current, 1 refused with nothing changed, 3 rolled back. An update.json written by an older installer lacks `AgentConfigurationFile`; this control plane then answers the archive download with HTTP 401 and the update installs nothing. Run `sudo bash upgrade-linux.sh` from a newer extracted package, which adds the property (see "In-place upgrade"), or add `"AgentConfigurationFile":"/etc/clm-agent/agent.json"` to the root-owned /etc/clm-agent-update/update.json yourself (keep it owned by root and not writable by others). Inspect with `journalctl -u clm-agent-update.service`. Self-update runs on Linux; on Windows `upgrade-windows.ps1` checks the same signed manifest (see "In-place upgrade"). You can also verify the signed ZIP yourself (PowerShell 7: import the PEM key with `[System.Security.Cryptography.ECDsa]::Create()` and `ImportFromPem`, then `VerifyData` with SHA-384 and `Rfc3279DerSequence`, or use OpenSSL). An updated release directory still matches the package `SHA256SUMS`. Publisher code signing is described in "Package verification" and "Package production".

25.1 In-place upgrade

A newer version is installed over an enrolled agent without a new enrollment code; configuration, identity and state stay as they are. Where a step below needs a newer package on the host, download its archive through the signed-in console (the download link in Agents and destinations or SSH automation) and copy it to the host, because archives are served only to an authorized request (see "Package downloads"); checksums, signatures and release manifests are public.

- **Linux, signed updates enabled** (installed with the release key): the daily timer upgrades automatically; `sudo bash upgrade-linux.sh` (in a newer extracted package) checks now. It adds `AgentConfigurationFile` to an existing update.json that lacks it, so installations made by an older installer can download archives again. It also lets the discovery service read the host's interface networks (`AF_NETLINK`) when the service was installed by an installer before 1.0.1.
- **Linux, installed without the release key**: get release-signing.pub and its fingerprint from your operator through a trusted channel and run `sudo bash upgrade-linux.sh ./release-signing.pub` from a newer extracted package. It pins the key, writes /etc/clm-agent-update exactly as the installer would (including `"AgentConfigurationFile":"/etc/clm-agent/agent.json"`, so the update presents this agent's client certificate), enables `clm-agent-update.timer`, prints the key fingerprint and runs `Clm.Agent update` at once. A different key than an already pinned one is refused. It needs a signed release (`.release.json.sig`); without a published key there is none and the upgrade stops with that reason.
- **Windows, download by the agent** (recommended with signed releases): in an administrator PowerShell run `powershell -NoProfile -ExecutionPolicy Bypass -File "$env:ProgramData\CLM-Agent\upgrade-windows.ps1" -Download -PublisherSha256 <PUBLISHER-SHA256> -ReleaseKeyFile .\release-signing.pub` (later upgrades may omit `-ReleaseKeyFile`, the key is pinned on first use). The installed script lets `Clm.Agent.exe download-release` fetch `clm-agent-win-x64.release.json`, its `.sig` and the archive from the agent address with this agent's client certificate (no console download, no copying), checks the archive size and SHA-256 against the manifest and then performs every check and step of the next item. Without a release key (`-ReleaseKeyFile` or a pinned one) `-Download` is refused. A successful upgrade with a release key also registers the daily update task `CERTYX Agent Update` (see Windows step 4), so later releases arrive without a manual run.
- **Windows**: the archive `clm-agent-win-x64.zip` is served only to an authorized request (see "Package downloads"), and the host is already enrolled, so no enrollment code applies: download the ZIP through the signed-in console (the download link in Agents and destinations or SSH automation) and copy it to the host. `clm-agent-win-x64.release.json` and `clm-agent-win-x64.release.json.sig` are public; download them from `/downloads/` of the agent address directly. Extract the ZIP and, in an administrator PowerShell in the extracted directory, run `.\upgrade-windows.ps1 -Archive ..\clm-agent-win-x64.zip -PublisherSha256 <PUBLISHER-SHA256> -ReleaseKeyFile .\release-signing.pub` (the key is pinned in `C:\ProgramData\CLM-Agent-Update` on first use; later upgrades may omit `-ReleaseKeyFile`). The script extracts the archive again into the protected program directory, verifies the publisher signature, the signed `SHA256SUMS` and the Authenticode signatures (as the installer does), then `Clm.Agent.exe verify-release` checks the manifest signature against the pinned key, the archive size and SHA-256 and that the version is newer. Only then it stops the tasks `CLM Discovery Agent` and `CERTYX SSH Workload`, moves the replaced program files to `C:\ProgramData\CLM-Agent\previous`, copies the new files, moves an identity from older installations into `identity\` (`Clm.Agent.exe migrate-identity`), restarts the tasks and requires `Clm.Agent.exe status` to succeed within 30 seconds. Otherwise it restores the previous files, restarts them and exits with code 3; a refused check exits with 1 and changes nothing. Without a published release key the script refuses unless `-AllowUnsignedDevelopment` is given (development only: publisher signature and version checks only).

Windows identity rotation before this version could not stage the next identity, because the program directory is read-only for LOCAL SERVICE: rotation then stops with "cannot write next to its identity". Run `upgrade-windows.ps1`, or as administrator `Clm.Agent.exe migrate-identity C:\ProgramData\CLM-Agent\agent.json` followed by `icacls C:\ProgramData\CLM-Agent\identity /grant *S-1-5-19:(OI)(CI)M`, and rotation continues.

26 Command-line checks

`Clm.Agent check <https-origin>` (before enrollment, no identity needed) and `Clm.Agent status <config-file>` (after enrollment, as the agent account; on Linux `sudo clm-agent status`) print one `[ OK ]`, `[WARN]` or `[FAIL]` line per step and end with `Result:`; exit code 0 means working, 1 a failed step, 2 wrong usage. They print host names, addresses, certificate subjects, the public certificate fingerprint and contact times, never tokens, passwords or keys.

Commands of `Clm.Agent` (`Clm.Agent.exe` on Windows); the first word selects the command:

- `--version`: prints the release version.
- `check <https-origin>` and `status <config-file>`: the checks above.
- `enroll <https-origin> <organization-id> <token-file> <config-file>`: one-time enrollment (see "Manual / container execution").
- `scan <config-file> [--once]`: network discovery; this is what clm-discovery.service and the Windows task run.
- `update <update.json>`: installs a newer signed release (Linux, see "Signed updates").
- `verify-release <release-signing.pub> <release.json> <release.json.sig> <archive> <rid> [<installed-version>]`: the offline check of a downloaded signed release (`VERIFIED`, exit 0, or `REFUSED: <reason>`, exit 1); used by `upgrade-windows.ps1`.
- `migrate-identity <config-file>`: moves a Windows identity from the program directory into `identity\` (see "In-place upgrade").
- `rotate-identity <config-file>`: requests an immediate identity rotation; the running discovery service or SSH workload replaces the key and certificate at its next identity check (see "Revocation and recovery").
- `download-release <config-file> <rid> <directory>`: downloads the published release manifest, its signature and the archive with the agent's client certificate into an existing directory; nothing is trusted or installed (used by `upgrade-windows.ps1 -Download`).
- `--log-file <path>` (with any command): writes the output to a time-stamped log file of at most 5 MB (plus `<path>.1`); the Windows tasks use it.

- `<config-file> [--once]`: the certificate lifecycle agent (any argument list not matching another command).
- `verify-endpoints <config-file> [--once]`: the independent endpoint verifier.
- `verifier add-target <agent-config> <verifier-directory> <target-file>` and `verifier remove-target <verifier-directory> <target-id>`: write the endpoint verifier configuration (used by `configure-verifier-linux.sh` and `configure-verifier-windows.ps1`).
- `ssh add-renewal-profile <ssh-directory> <profile-file>`: writes `renewal.json` of the standalone local renewal (used by `enable-renewal`).
- `secret-check <config-file>`: preflight of customer secret references (CUSTOMER-VAULTS.en.md).
- `relay-helper`: prints the relay helper of this signed agent for installation on hosts an SSH relay deploys to (see "SSH relay").
- `relay-check <lifecycle-config> [destination-id]`: checks the SSH relay destinations without changing a certificate: host key pin, authentication, helper version, store directory, the service listening on the TLS port with the suggested connector, and the configuration check (exit 0 ready, 1 a failed step, 2 wrong usage).
- `aws-plan`, `aws-adopt`, `gcp-adopt`, `azure-adopt` and `digicert-collect`: cloud and DigiCert preparation (AWS-LIFECYCLE.md, GCP-LIFECYCLE.md, AZURE-LIFECYCLE.md, CA-CONNECTORS.md).
- `ssh manage|renew|monitor-krl|publish-krl|sync-krl <config-file> [--once]`, `ssh apply-krl <consumer-config> <envelope-file>` and the local authority commands `ssh issue|revoke|list|export-krl|deploy-host|deploy-user` (Unix only, SSH-LIFECYCLE.md).

The Linux installer adds the helper `/usr/local/bin/clm-agent`: `sudo clm-agent status`, `sudo clm-agent check`, `sudo clm-agent log [lines]` and `clm-agent version`. On Windows call `Clm.Agent.exe` in `C:\ProgramData\CLM-Agent` directly from an administrator PowerShell.

Wrong argument counts and unknown options stop with `Usage:` and exit code 2 before any configuration is read or any directory is created: `secret-check`, `scan <config> [--once]`, `enroll`, `Clm.Agent <config> [--once]` and the SSH subcommands with their own configuration (`manage`, `renew`, `monitor-krl`, `publish-krl`, `sync-krl`, `apply-krl`). Enrollment requires the token file to be a regular, owner-only file without symbolic links in its path; this is checked before the token is read or the configuration directory is created, and enrollment responses are limited to 64 KiB. A lifecycle configuration missing a required member (control plane, client certificate, state directory, destinations with ID and domains, certificate password source) stops with `InvalidDataException`. The discovery checkpoint is written owner-only and durably. Commands started by the agent (nginx, service reload, ssh-keyscan, ssh probe) never read the agent's standard input; the SSH relay writes file contents and keystore passwords only to the standard input of its own ssh session.

27 Windows host prerequisites

Before installation, run `powershell -NoProfile -File .\check-windows.ps1` from an administrator PowerShell in the extracted package. It writes `certyx-windows-preflight.json` containing only OS/version/architecture, PowerShell version, administrator status and task/SSH availability. No passwords, tokens, certificate contents, hostnames or user identities are collected. The report only lists prerequisites; it does not install the agent or verify the package. Confirm the approved package hash/signature before executing any package content. The installer requires an explicit HTTPS origin, refuses linked token/installation paths and removes its local copied enrollment token even if enrollment fails. The original token file remains under your control; delete it after use.