MIRAKI LABS · OPC CONNECT X

User Manual

Full reference for configuration, bridges, historian, and security

Document version 1.8.0

OPC Connect X is a Windows connector that moves industrial data between OPC servers, records it, and forwards it to IT systems. This manual is the complete reference: every screen, every bridge type, and the settings behind them. If you only need to get running, read the *Quick Start Guide* first — this document assumes the app is installed and licensed.

1. Concepts

Six ideas explain the whole product. Everything in the interface is one of these.

Endpoint — a saved OPC server: an OPC UA endpoint URL with its security and credentials, or an OPC DA server identified by ProgID on a host. Endpoints are registered once and reused by every collector and bridge.

Collector — a session that reads tags from one endpoint continuously (UA subscriptions, DA group polling) and archives the values. Collectors are the source of everything the Historian and Trends show.

Internal bridge — tag routing **within one server**: value on tag A is written to tag B on the same endpoint. Used for intercom and mirroring inside a single PLC or DA server.

External bridge — transfer **out of a server**: to a different OPC server, or to an MQTT broker or REST endpoint. This is the classic north-bound path.

Historian — the local time-series store. Collected and bridged values land here; you query, downsample, chart and export from it.

Security (ACL) — allow/deny rules at tag level, enforced by the backend engine rather than by hiding buttons in the UI.

What you are allowed to start depends on your license; see the *License Guide*.

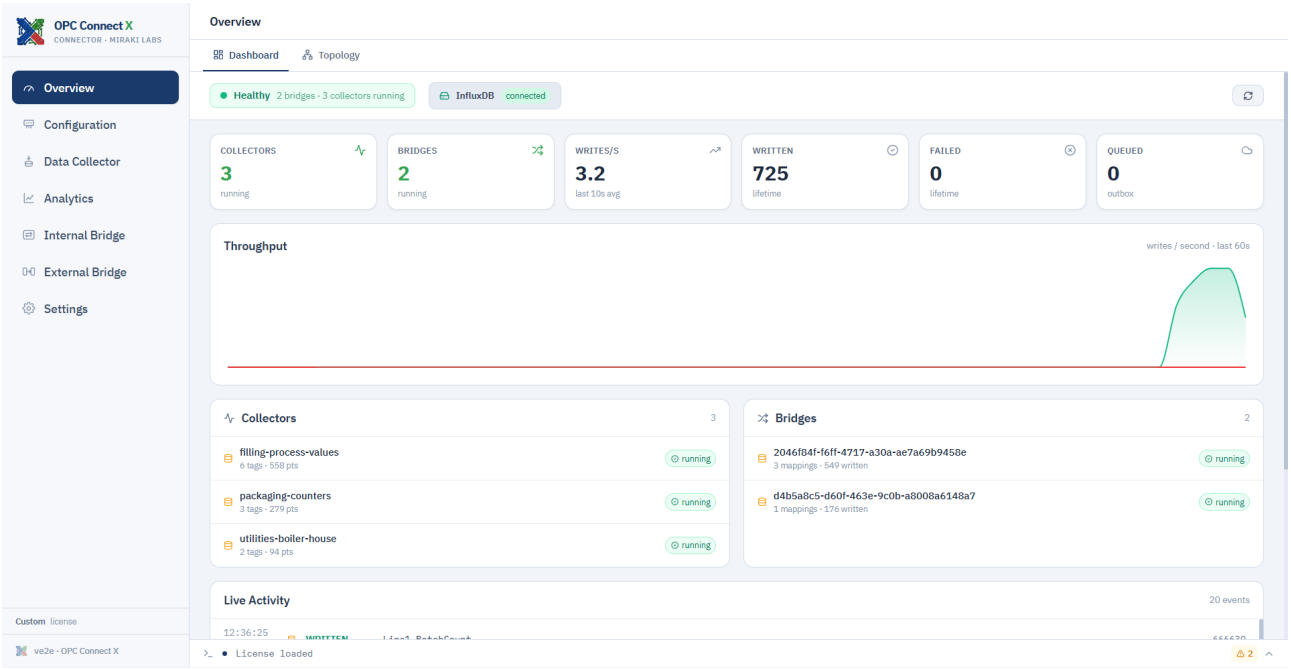
2. Getting oriented

2.1 Layout

The window is a single application with a left sidebar. The sidebar is grouped by protocol — **OPC UA** and **OPC DA** each have their own configuration, collector, bridge and analytics screens — plus shared sections for Historian, Trends, Diagnostics, Security and Settings. The installed version is shown in the sidebar.

Screens your license does not cover remain visible so you can see what the product offers; the actions inside them are refused with an explanation.

2.2 Dashboard

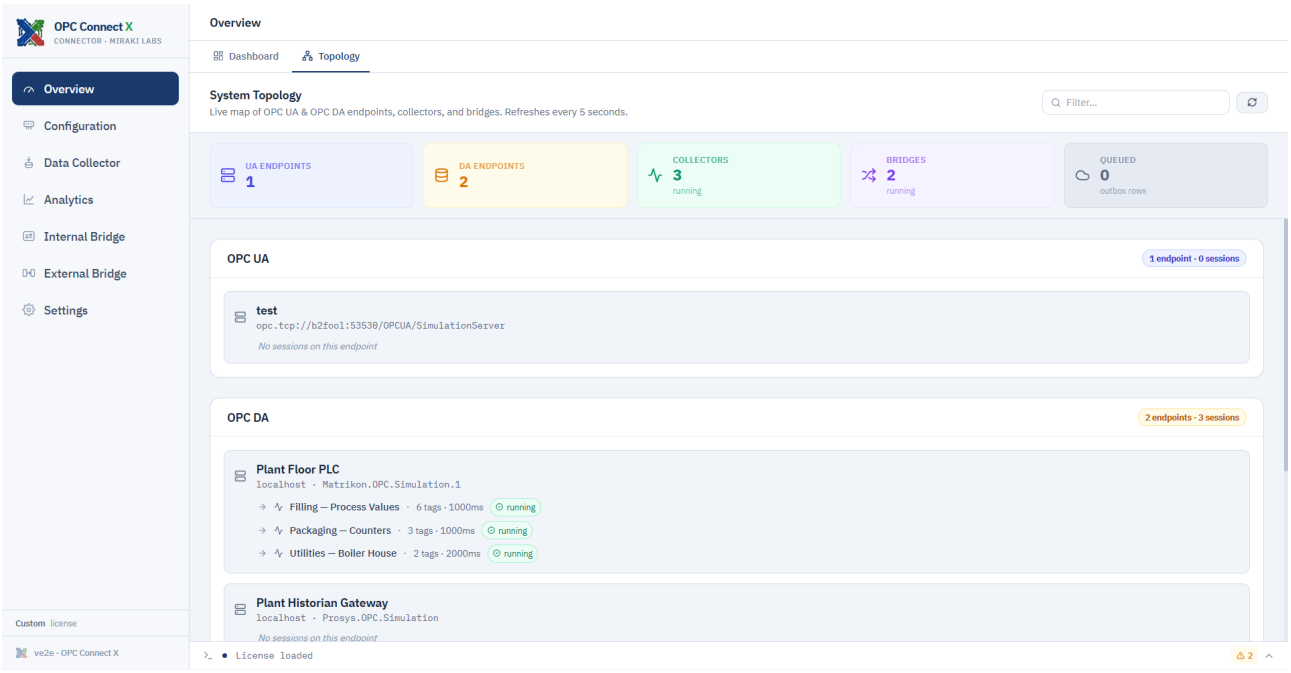


Live overview — every running collector and bridge with real throughput counters.

The Dashboard is the operational summary: which collectors and bridges are running, how many points each is moving, connection state per endpoint, and recent errors. Counters are live, straight from the engine — not a periodic snapshot.

Use it as the first stop after a restart: anything that failed to resume shows here with its reason.

2.3 Topology



System topology — servers, collectors and bridges as a live wired graph.

Topology draws the same information as a graph: servers as nodes, collectors and bridges as the wires between them. On a site with several DA servers and a couple of bridges this is the fastest way to confirm the data path is what you intended — and to spot the mapping someone added last month that still points at a decommissioned server.

2.4 Tray and window behaviour

Closing the window does **not** stop the engine. The application keeps running in the notification area so bridges continue; use the tray menu to restore the window or exit properly. For a machine that must run without anybody logged in, use service mode (chapter 10) instead.

3. Connecting to OPC UA servers

3.1 Registering an endpoint

OPC UA → **Configuration** → **Endpoints** holds the registry of UA servers.

To add one:

1. Enter the endpoint URL, for example `opc.tcp://192.168.1.50:4840`, and a friendly name.
2. Choose the **security policy** and **mode** — the pair the server actually offers. `None`/`None` is fine on a closed test bench; production usually wants `Basic256Sha256` with `Sign` or `SignAndEncrypt`.
3. Choose **authentication**: anonymous, username and password, or a client certificate.
4. Save, then **Test Connection**.

Credentials are stored encrypted in the configuration file, not in plain text.

3.2 Discovery

OPC UA → **Configuration** → **Discovery** queries a host for the endpoints it publishes, with their security policies and modes, so you can register the right one rather than guessing. Point it at a host or a discovery server URL and select from the result.

3.3 Certificates and trust

Secure connections need a mutual trust step, and it is the single most common reason a first connection fails.

- The app has its own client certificate; the **Certificates** screen shows it, its expiry, and lets you regenerate it. The app also raises an event as expiry approaches.
- On the first `Sign` or `SignAndEncrypt` connection, most servers reject the client and park its certificate in a rejected/quarantine folder. An operator must move it to the trusted list on the **server**, after which the connection succeeds.
- If the server's own certificate is not yet trusted here, accept it when prompted.

A connection that fails with a security-check error almost always means one of those two trust steps is still pending — not a wrong password.

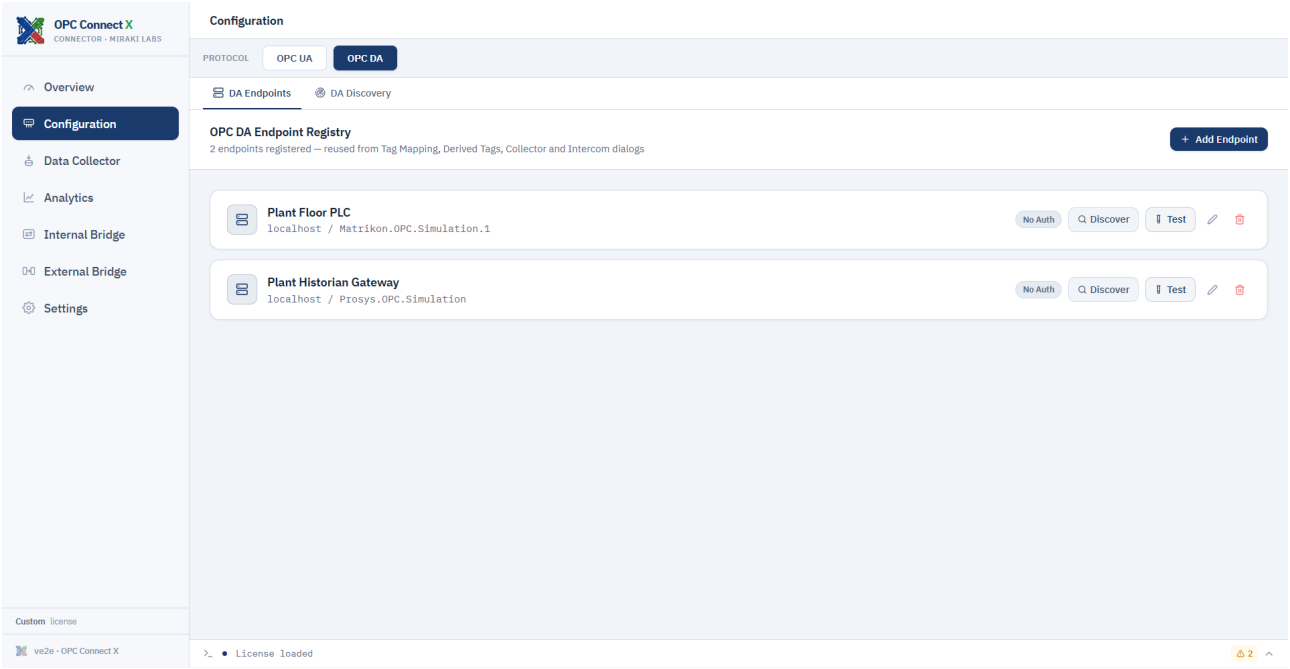
3.4 Tag mapping and browsing

OPC UA → **Configuration** → **Tag Mapping** is where source node IDs are paired with their destinations for bridging. Browsing is done from the address space tree of the registered endpoint; very large address spaces are rendered in capped, scrollable pages so the interface stays responsive on servers with tens of thousands of nodes.

4. Connecting to OPC DA (Classic) servers

OPC DA predates OPC UA and runs on COM/DCOM. Most DA servers and their proxy/stub DLLs are 32-bit, so OPC Connect X talks to them through `opcda-bridge.exe`, a small 32-bit helper process that must sit in the same folder as the main executable. Everything below fails, with a clear message, if that file is missing.

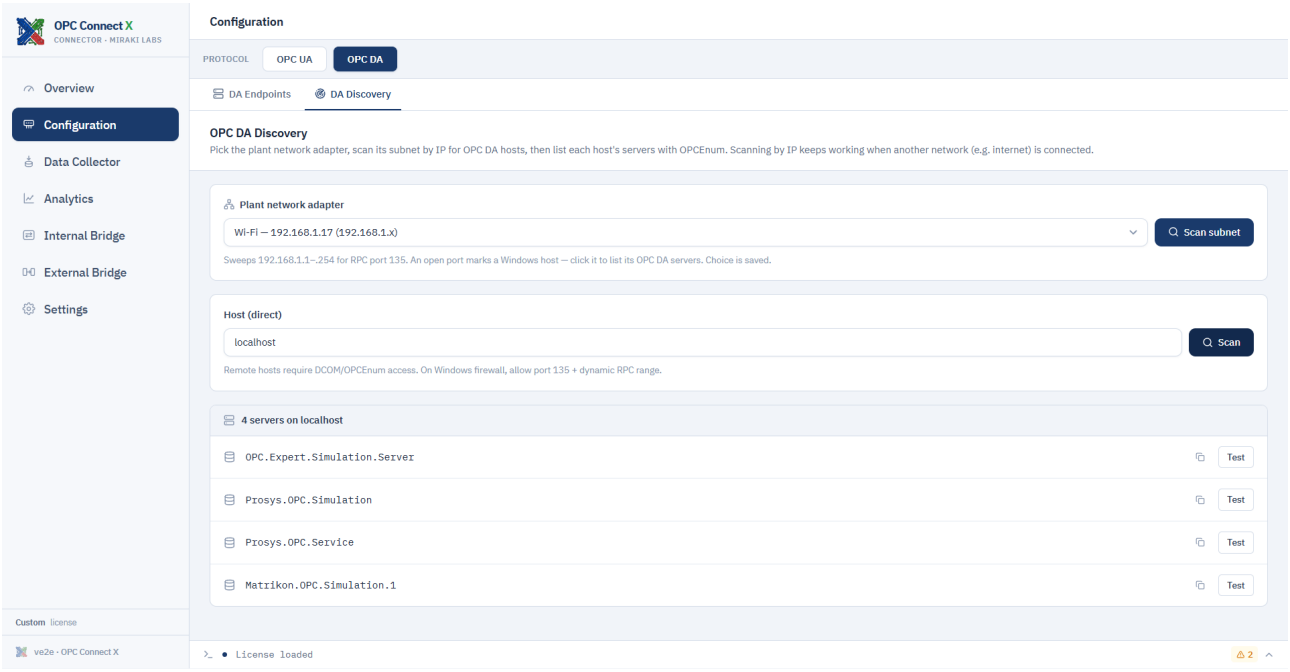
4.1 Endpoint registry



OPC DA endpoint registry — every classic server the plant talks to, in one place.

OPC DA → **Configuration** → **Endpoints** lists every Classic server the installation talks to: friendly name, host, ProgID, and current reachability. Register a server here once; collectors and bridges then refer to it by name.

4.2 Discovery



Discovery — enumerates the OPC DA servers registered on a host over DCOM.

Discover asks a host — `localhost` or a remote machine — which OPC DA servers are registered on it, and returns their ProgIDs. On a remote host this goes over DCOM, so the usual DCOM prerequisites apply: the account must be accepted by the remote machine, and the firewall must allow RPC.

For remote servers the connection resolves the server's CLSID on the remote machine rather than relying on this machine's registry, so a server that is not registered locally still connects.

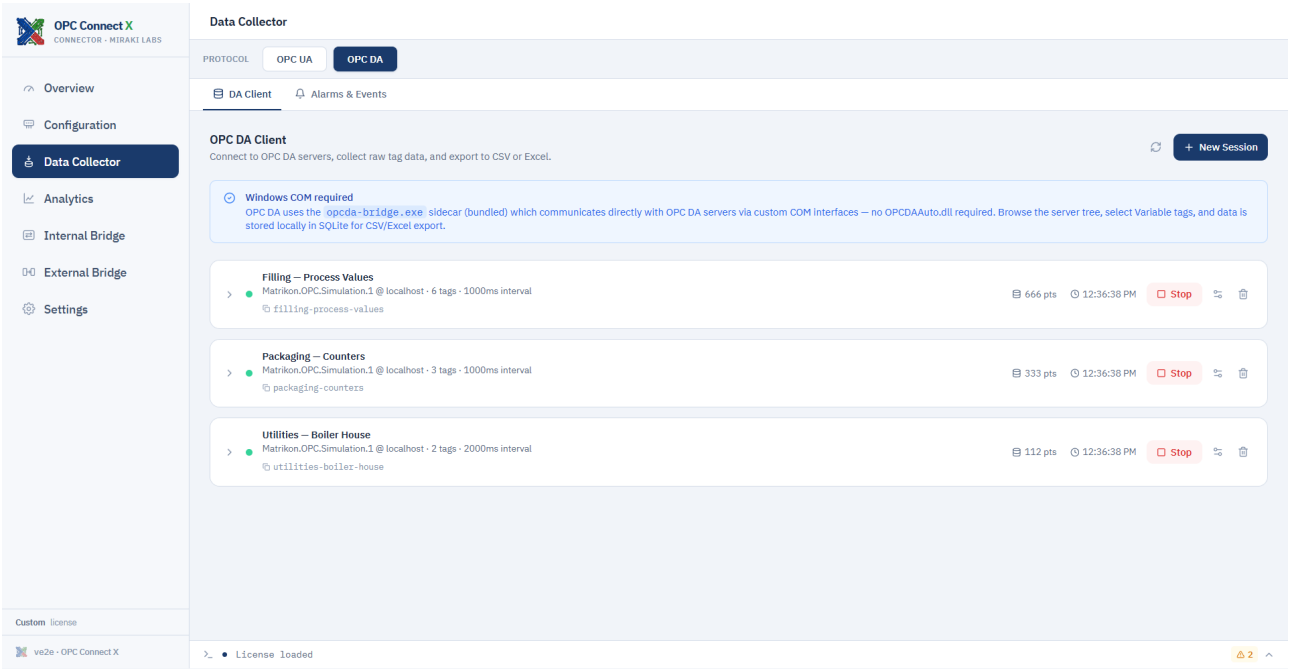
4.3 When DA will not connect

Classic OPC failures are nearly always environment, not configuration. Use the **Diagnostics** screen (chapter 11) — it walks the connection step by step and names the stage that failed. The most common causes:

Symptom	Cause
No servers listed on a remote host	DCOM permissions or firewall on that host
Access denied	The account running OPC Connect X (or the service account) is not accepted by the server machine
Interface-not-supported errors	Wrong-bitness OPC proxy/stub DLLs registered on the server machine
Server appears then drops	DCOM auto-relaunch of a server that crashed; check the server's own logs

5. Collecting data

5.1 Collector sessions



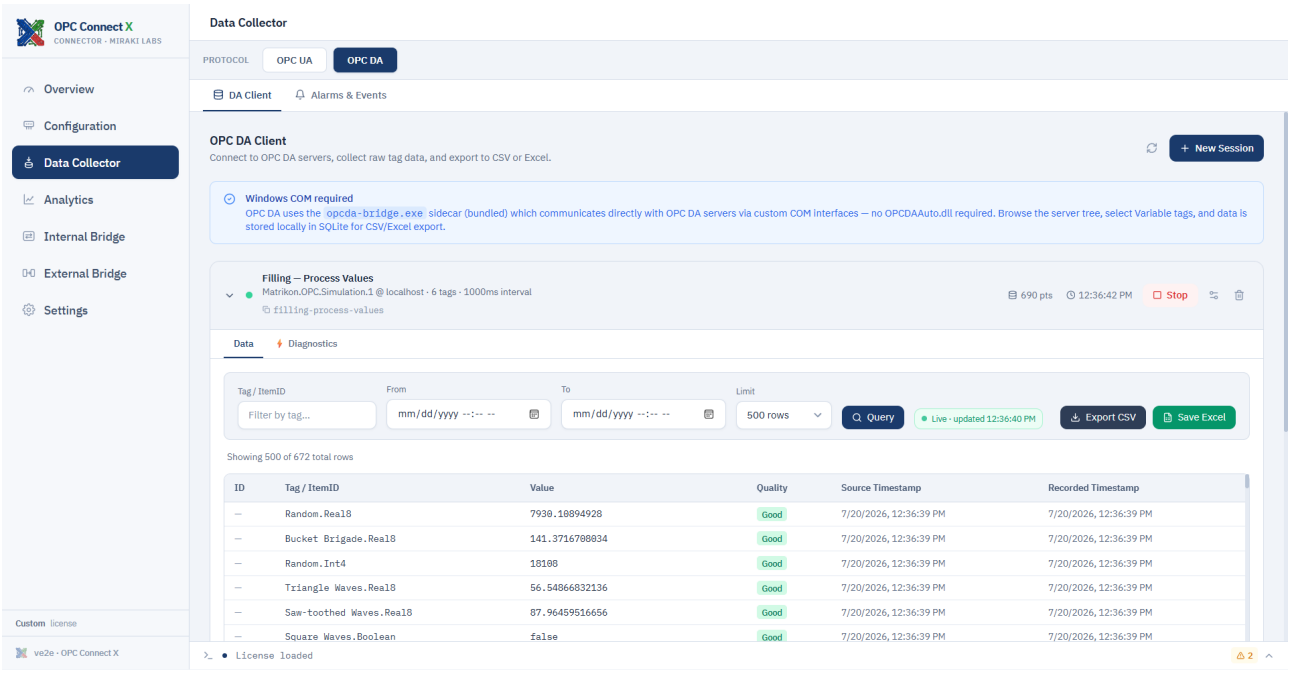
OPC DA collector sessions polling real tags, with per-session point counts.

A **collector session** binds one endpoint to a set of tags and an update rate. Both protocols work the same way from the operator's point of view:

1. Create a session and pick the endpoint.
2. Browse the server and select tags. Filter by name to cut a large tree down.
3. Set the rate — a subscription publishing interval for UA, a group update rate for DA.
4. Start the session.

Each running session shows its point count, update rate and connection state. Sessions survive restart: they are stored in the database and resumed automatically (also in service mode).

5.2 Live values



Per-tag live values, quality and update counts straight off the wire.

Inside a session, each tag shows its current value, data type, OPC quality and how many updates it has received. This view is the ground truth when someone asks whether a tag is actually moving — it is the raw value as delivered by the server, before any transform.

Timestamps are normalised to UTC on the way in. Some DA servers send local time in a field defined as UTC; the collector corrects for that so history does not end up stamped in the future.

5.3 Export

Session data exports to **CSV or Excel** for offline analysis or hand-off to another team. Historian exports (chapter 8) cover longer ranges with downsampling.

5.4 Alarms & Events

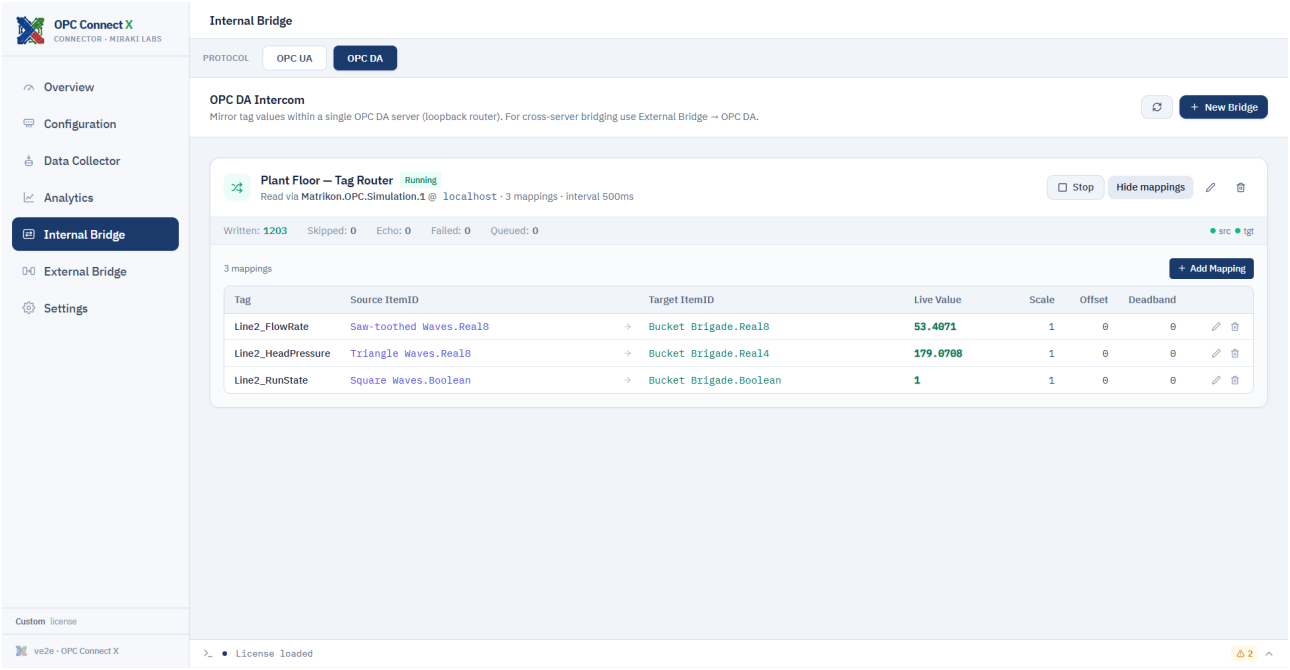
Where the license includes it, **Alarms & Events** subscribes to an OPC AE server, records alarm and condition transitions with their timestamps and severities, and lets an operator acknowledge active conditions from the app. The record is stored alongside the historian data so an event can be lined up against the process values around it.

6. Bridges

Four bridge shapes exist. Choosing the right one is most of the work.

Shape	Screen	Typical use
UA → UA, same server	OPC UA Internal Bridge	Intercom / mirroring inside one UA server
DA → DA, same server	OPC DA Internal Bridge	Classic intercom within one server
DA → DA, different servers	External Bridge (DA)	Move a value from PLC-A's server to PLC-B's server
UA or DA → MQTT / REST	External Bridge (UA/DA)	North-bound to a broker, cloud or REST API

6.1 Internal bridge (same server)



DA Intercom — same-server tag routing with echo suppression, mappings expanded.

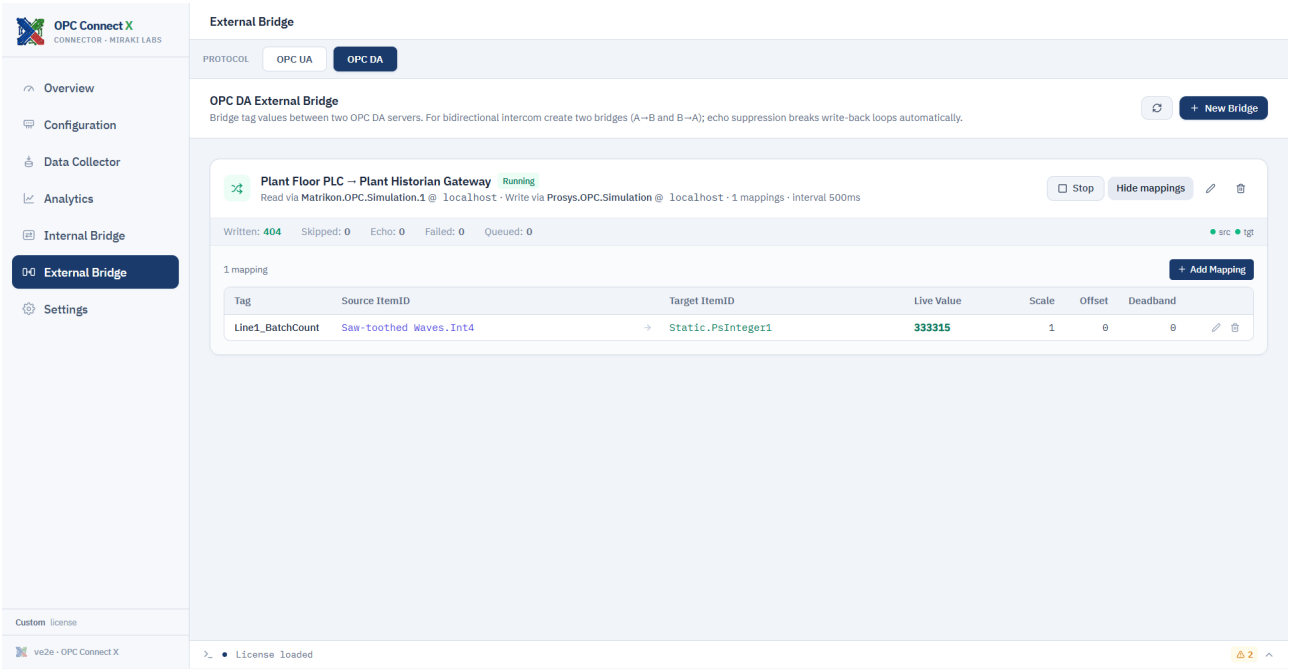
An internal bridge routes values between tags on **one** endpoint. Create the bridge, then add mappings: source tag, target tag, and per-mapping options.

Per mapping you can set:

- **Deadband** — only write when the value moved by more than a threshold, to keep a noisy analogue from hammering the target.
- **Echo suppression** — see 6.4.
- **Data-type handling** — the engine checks the target's type and refuses a mapping that cannot be represented, rather than writing a silently truncated value.

> A mapping added to an **already running** bridge is not pushed into the live engine by creating it alone. Apply the update to the running bridge (or restart it) for the new mapping to start moving data.

6.2 External bridge, server to server



Cross-server DA bridge running live between two different OPC DA servers.

A cross-server DA bridge copies a tag from a **source** server onto a tag on a **different target** server. Per bridge you configure:

- Source endpoint and target endpoint.
- A poll rate for the source.
- One or more tag pairs, each optionally with a **scale and offset** transform (`out = in × scale + offset`), so engineering units can be converted in flight.
- An optional label per mapping for readability in the monitor.

The monitor view shows live counters per bridge: **Written**, **Skipped**, **Queued**, **Failed**, and **Echo** — enough to tell "nothing is moving" apart from "everything is moving and being rejected at the target".

6.3 External bridge to MQTT / REST

The same source side, a different target: values are published to an **MQTT broker** (topic, QoS, credentials, optional TLS) or **POSTed to a REST endpoint** (URL, headers, auth).

Because IT links drop more often than plant links, this path has **store-and-forward**: when the target is unreachable, values are queued to disk and drained in order once the connection returns, instead of being lost. The queued counter on the monitor shows the backlog.

6.4 Echo suppression

Two bridges pointing at each other (A → B and B → A) would otherwise loop forever: the value written to B comes back as a change on B and is written to A, and so on. An **echo tracker** remembers what the engine itself just wrote and suppresses the reflection. The Echo counter on the monitor tells you how many write-backs were recognised and dropped.

Turn echo suppression on for any bidirectional mesh. Leave it on even for one-way bridges where the target is also collected — it costs nothing and prevents a surprise the day someone adds the reverse direction.

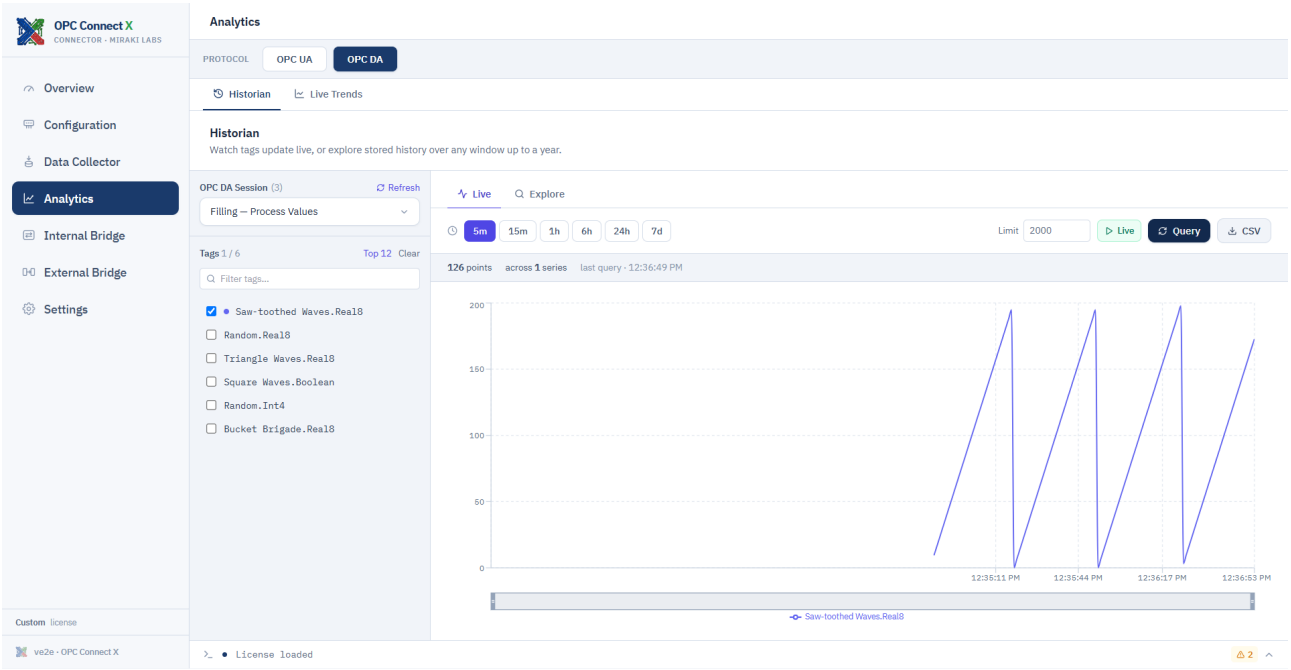
6.5 Failover

A bridge can be given a **primary** and a **secondary** source. The engine runs a small state machine: primary active, primary failed, secondary active, and a periodic recovery test that promotes the primary back once it proves healthy again. The current state is visible on the bridge, so an operator can see that a bridge is running *on its backup* rather than merely "running".

6.6 Derived tags

Derived Tags (UA and DA each have their own screen) compute new values from existing ones with a formula — arithmetic, comparisons and functions over other tags. A derived tag behaves like any other tag downstream: it can be historised, trended, bridged or published. Typical uses are unit conversion, simple totalisers, and combining several raw signals into one health value.

7. Historian



Historian — query and export recorded tag history, downsampled on demand.

The Historian is the local time-series store for collected and bridged values.

Querying. Pick a session or tag set, choose a time range, and read the values back. Long ranges are **downsampled on demand** — the query returns a representative series rather than trying to render millions of raw points.

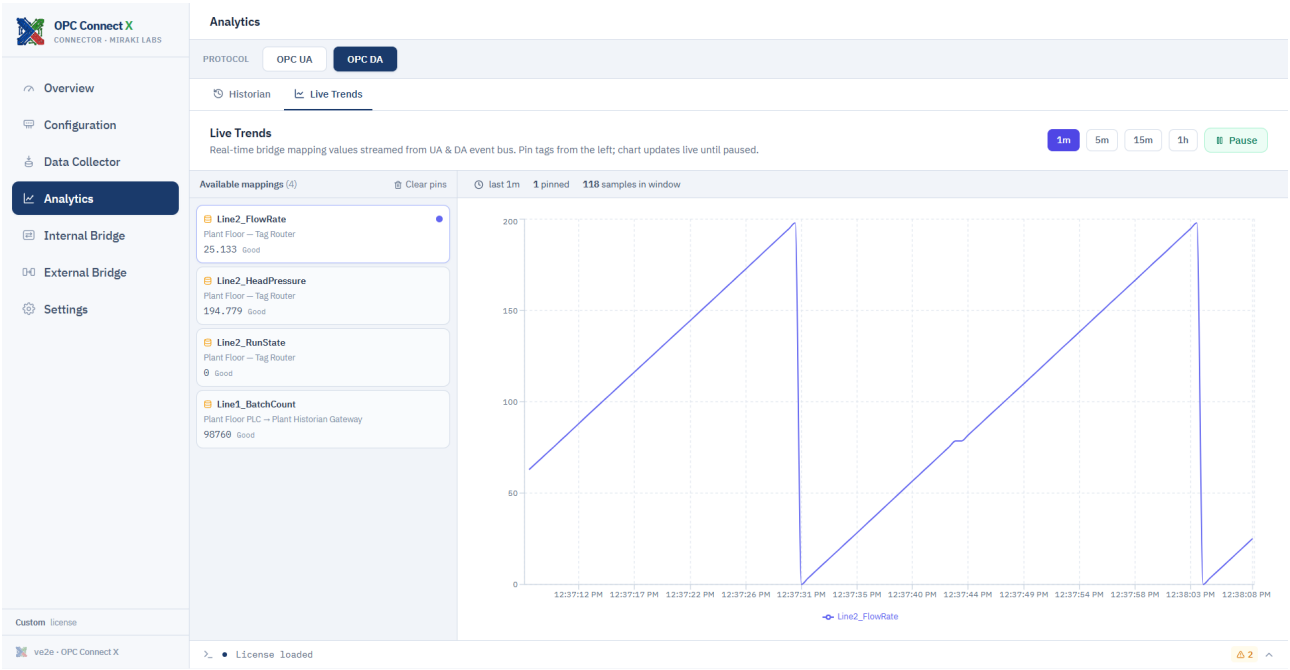
Aggregation. Values can be returned aggregated over an interval (for example averages per minute) instead of raw samples, which is what you want for reports and for charting a week of data.

Export. Any query result exports to CSV.

Storage. By default history lives in the embedded SQLite database in the data directory — no server to install. Where a site already runs **InfluxDB 1.8**, it can be enabled in Settings as the primary time-series store instead; the app ships the pieces needed to run it locally.

Retention. History grows with tag count and rate. Plan disk accordingly, and keep the data directory on a drive with room — a few hundred tags at one-second rates is gigabytes per month.

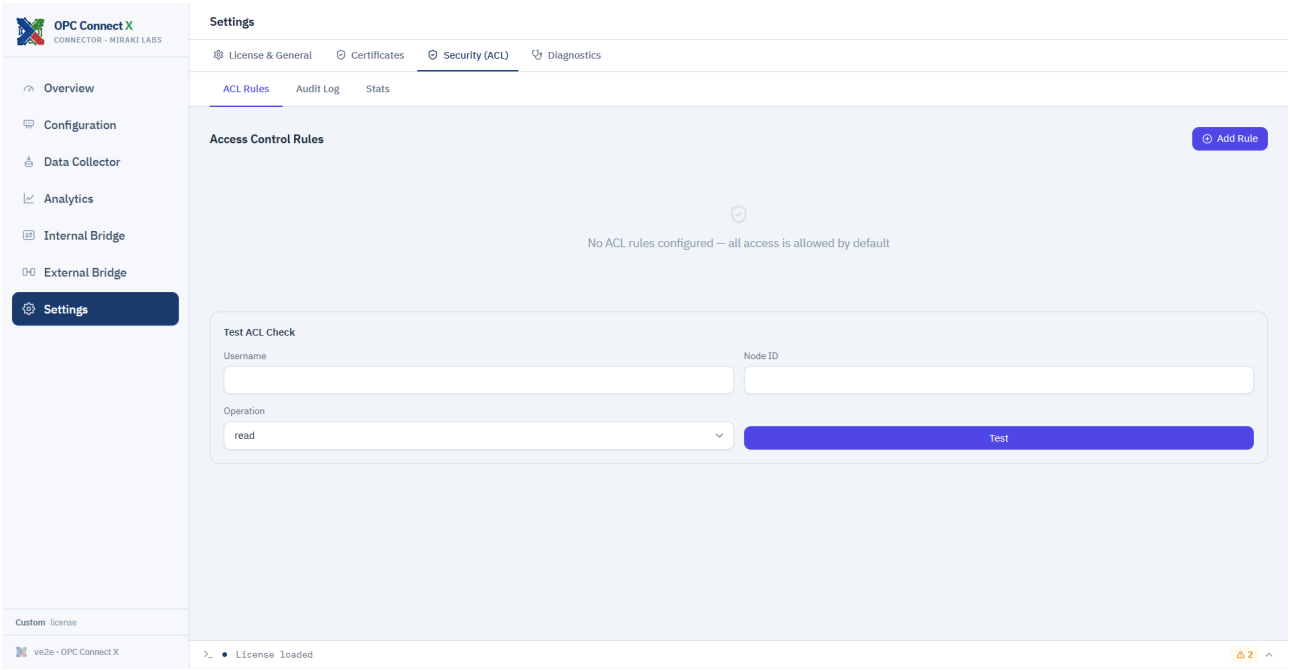
8. Live Trends



Live trends — streaming values from running bridge mappings.

Trends charts values as they arrive. Pin tags to a chart, use multiple Y-axes for signals with different ranges, and switch between live streaming and historical playback of a past window. It is the screen to keep open during commissioning: you can watch a source and its bridged target on one chart and see the transfer happen, including the delay introduced by the poll rate.

9. Security



Tag-level access control — allow/deny rules enforced in the backend.

The **Security** screen holds an access-control list at **tag level**: allow or deny rules that decide which tags may be read and, more importantly, which may be **written**.

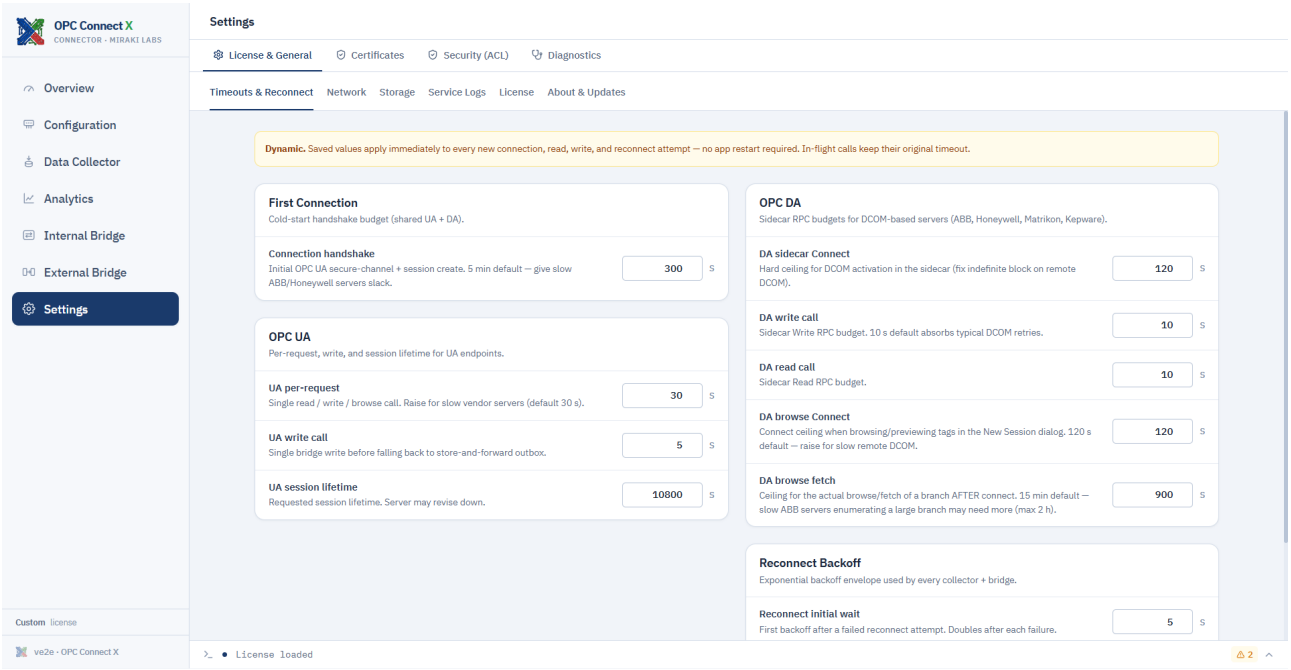
Two properties matter:

- Rules are enforced in the **backend engine**. A denied write is refused where the write happens, not by disabling a button — so it also holds in service mode, where there is no UI at all.
- Denial is explicit and logged, so a blocked write shows up in the audit log rather than vanishing.

Use deny rules on anything that must never be written from this connector — setpoints, mode words, interlock bits — even if no current mapping targets them. It costs nothing today and stops tomorrow's mistake.

All meaningful actions are written to the structured audit log in the logs directory.

10. Settings



Settings — timeouts, storage, service logs and licensing.

License File / License Status. Import a `license.lic`, and read back tier, license ID, expiry, days remaining and the exact feature set granted. Your **Machine ID** is shown here — it is what a license is bound to. See the *License Guide*.

Bridge Pipeline Settings. Engine-level behaviour, including **auto-start saved bridges on launch**. Turn auto-start off on a commissioning machine where you want to decide manually what runs; leave it on for any production installation, and always for service mode.

Time-Series Storage. Choose the store: the embedded database, or **InfluxDB 1.8** with host, port and optional credentials.

Plant network adapter. On a machine with several NICs — one on the plant network, one on the office LAN — pin the adapter used for OPC traffic so connections do not take the wrong route.

Service Logs. A live view of the application log with a level filter (DEBUG / INFO / WARN / ERROR). This is the first place to look when something failed while nobody was watching; the same content is on disk as JSON.

About. Installed version and build information — quote this in support requests.

11. Running as a Windows Service

For unattended operation, the same executable runs headless as a Windows service. In service mode there is no window: the engine loads the configuration, resumes every saved bridge and collector from the database, and logs to the usual folder.

Run PowerShell **as Administrator**:

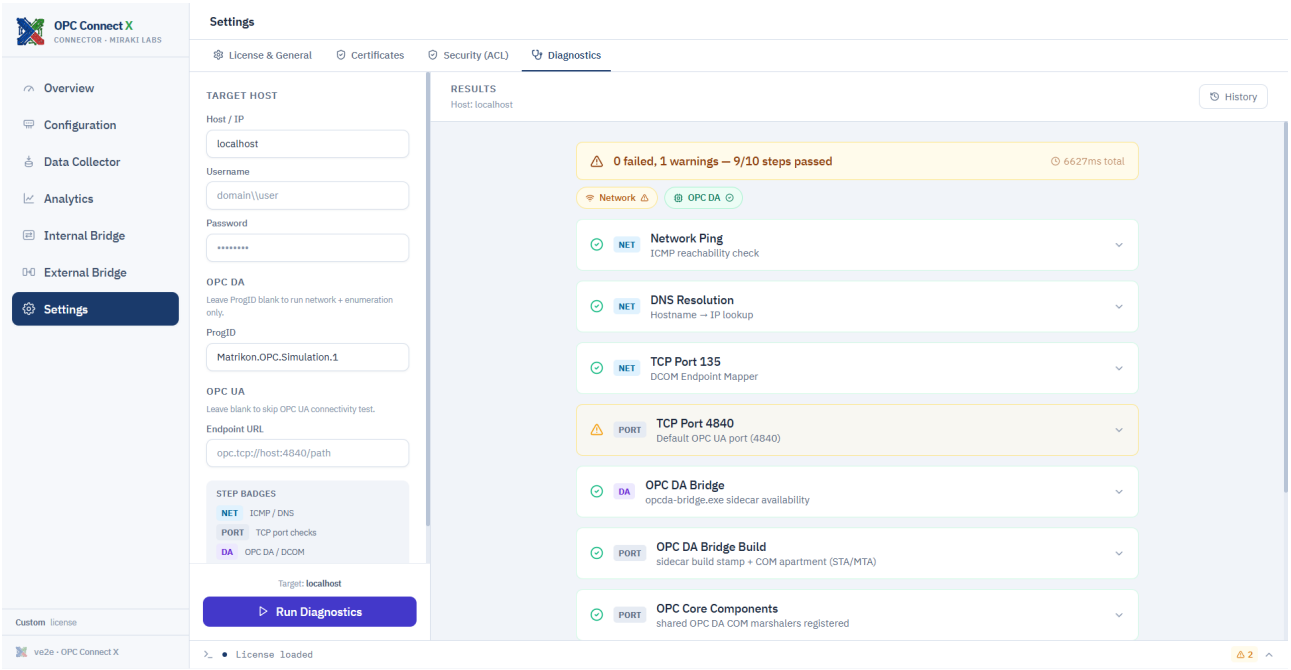
```
sc.exe create "OPCConnectX" binPath= "C:\OPCConnectX\OPCConnectX.exe service" DisplayName= "OPC Connect X Br
sc.exe start OPCConnectX
sc.exe stop OPCConnectX
sc.exe delete OPCConnectX
```

Three things to get right:

1. **Service account.** OPC DA over DCOM authenticates as the account the service runs under. `LocalSystem` is usually refused by a remote DA server. Give the service a domain or local account that the DA machine accepts — the same account you would log in with to test the connection by hand.
2. **License.** The license file is read from the data directory. Confirm the service account can read `%APPDATA%\OPCConnectX\` — its `%APPDATA%` is not your `%APPDATA%`.
3. **Auto-start.** Leave "auto-start saved bridges" enabled, or the service will start and do nothing.

Configure and test everything in the GUI first, then stop the GUI and start the service. Running both against the same data directory at once is not a supported configuration.

12. Diagnostics and troubleshooting



Built-in DCOM/OPC diagnostics — step-by-step probe of a real server.

The **Diagnostics** screen probes a real server step by step — resolve the host, enumerate servers, activate the COM object, add a group, read a tag — and reports which step failed and with what code. For Classic OPC, that is far more useful than a generic "connection failed", because the failing step points straight at the cause: name resolution, DCOM permissions, bitness, or the server itself.

Diagnostics also exposes per-tag quality and staleness and reconnect counters, so a link that works but keeps flapping is visible as such.

12.1 Common problems

Symptom	Where to look
Blank/white window on launch	WebView2 Runtime missing
No DA servers found at all	`opcda-bridge.exe` missing from the app folder, or .NET Framework 4.8 not installed
Remote DA fails, local works	DCOM permissions / firewall on the remote host; run Diagnostics against it
DA works in GUI, fails as a service	Service account not accepted by the DA server — see chapter 11
UA connects with `None` but not with `Sign`	Client certificate not yet trusted on the server
Bridge running, target never updates	Check Skipped/Failed counters; then check Security deny rules and the target tag's data type
A new mapping does nothing	Bridge is running with the old mapping set — apply the update or restart it
Everything blocked after a date change	Clock guard: the system clock moved backwards — see the *License Guide*

12.2 Logs

Structured JSON logs, one file per day:

```
%APPDATA%\OPCConnectX\logs\opcconnectx-<YYYY-MM-DD>.json.log
```

They cover both GUI and service mode and are the right attachment for a support request. Raise the console level to DEBUG in Settings while reproducing a problem, then set it back.

13. Files, backup and upgrade

Everything the application owns lives in one directory:

Path	Contents
`%APPDATA%\OPCConnectX\config.json`	Endpoints and bridge configuration (sensitive fields encrypted)
`%APPDATA%\OPCConnectX\bridge_data.db`	SQLite: mappings, sessions, historian, logs
`%APPDATA%\OPCConnectX\license.lic`	License
`%APPDATA%\OPCConnectX\features.json`	Cached feature flags
`%APPDATA%\OPCConnectX\logs\`	Daily JSON logs

Backup = stop the app or service, copy that folder. **Restore** = put it back on a machine with a valid license for *that* machine.

Upgrade: install the new version over the old one; the database schema is migrated on first start. The installer package supports in-place update. Take a copy of the data directory before a major upgrade — it is the only irreplaceable part.

14. Support

- Sales and licensing: sales@miraki.in
- Include: installed version (Settings → About), your `license_id`, the log file covering the problem, and what you expected to happen.

Companion documents: **Quick Start Guide**, **License Guide**, **Release Notes**.