ToddySoft Connect OPC UA - Apache StreamPipes

An Apache StreamPipes **adapter** for OPC UA servers, built on the ToddySoft Connect OPC UA driver. It reads variables from any OPC UA server, chosen from the server's browsed address space. Every polling interval produces one event in StreamPipes with the selected tags' values.

The adapter comes as its own **extension service**: a container that runs next to your StreamPipes installation, registers with it, and appears in StreamPipes as the adapter *ToddySoft Connect OPC UA*.

Requirements

- **Apache StreamPipes 0.99**, installed with Docker (the standard installer). The service joins the installation's Docker network.
- **Linux on the machine running StreamPipes**, amd64 or arm64, with a route to the devices. Docker Desktop (macOS, Windows) runs containers in a virtual machine: adapters work with *Enter address*, but no search reaches the plant network.
- A **ToddySoft licence for OPC UA** for production use. Without one the service runs on its bundled trial licence until that expires.

Installation

Loading the image

The service is delivered as an image archive, one per architecture, in the share you received your licence and downloads from: `streampipes-opcua-<version>-<architecture>.tar.gz`, for `amd64` or `arm64`. Use the archive of the version you were sent and the architecture of the machine (`uname -m`: `x86_64` is amd64, `aarch64` is arm64).

```
docker load < streampipes-opcua-1.3.0-amd64.tar.gz
```

> **The images are not on Docker Hub or any public registry**, and the same goes for the discovery agent: every image reaches you through this one channel. We plan to move to a private registry with a pull account per customer, so that `docker pull` and updates work without archives; until then, load each new version the same way.

Running the service

Start the service in the Docker network of your StreamPipes installation (`spnet` for the standard installer; `docker network ls` shows it):

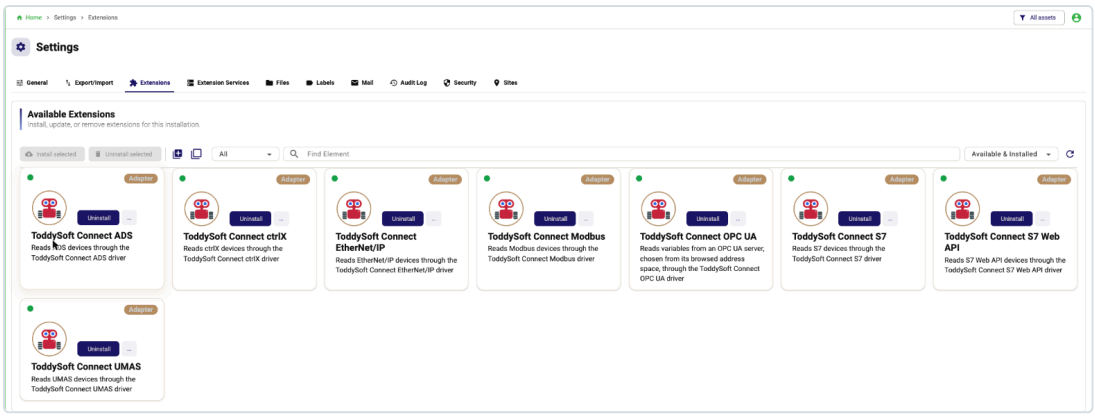
```
docker run -d --name toddysoft-opcua --restart unless-stopped \
  --network spnet \
  -e SP_HOST=toddysoft-opcua -e SP_PORT=8005 \
  -e SP_CLIENT_SECRET=<the SP_CLIENT_SECRET of your installation> \
  -e SP_PRIORITIZED_PROTOCOL=kafka \
  toddysoft/streampipes-opcua:1.3.0
```

- **`SP_HOST`** must be the container's own name: StreamPipes calls the service back by it.
- **`SP_CLIENT_SECRET`** must be the value the StreamPipes core runs with (the installer's `.env`). With another value the service cannot register.
- **`SP_PRIORITIZED_PROTOCOL`** is your installation's broker: `kafka` for the standard installer.
- Keep the service on the installation's network only. It reaches the core and the broker there by name; host networking would lose them. The devices are reached through the machine's routing, and the search through the discovery agent (see Reaching the devices).
- Files the adapter needs - a keystore, a trust store - are mounted with `-v`, for example `-v /opt/toddysoft/certs:/certs:ro`, and given in the form as `/certs/<file>`.

`docker logs toddysoft-opcua` should show the service registering with StreamPipes within a minute.

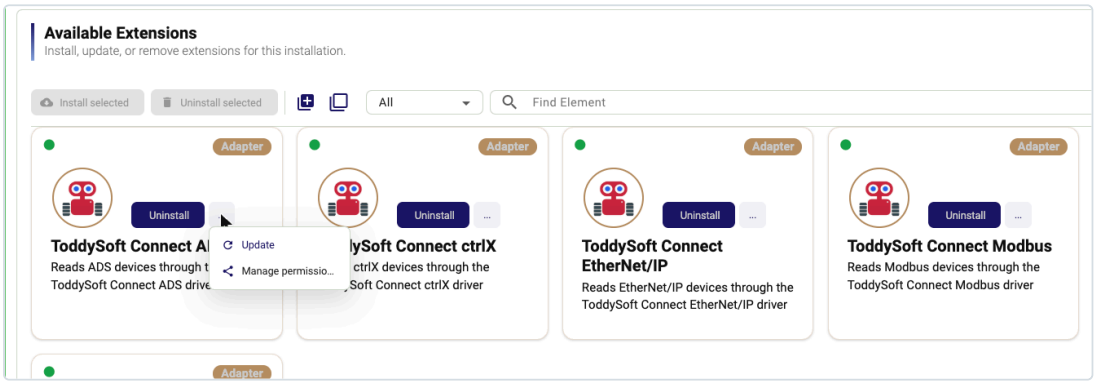
Installing the adapter in StreamPipes

Once the service has registered, install its adapter once: **Settings** → **Extensions**, find *ToddySoft Connect OPC UA* and press **Install**. It then appears under **Connect** → **New Adapter**.



Updating

1. Load the new archive and start the new version in place of the old container (`docker rm -f toddysoft-opcua`, then the `docker run` above with the new version).
2. **Update the installed adapter**: Settings → Extensions → the adapter's **"..."** → **Update**. StreamPipes keeps the adapter description it stored at installation, so without this step the form of the previous version is shown, and new settings are missing.

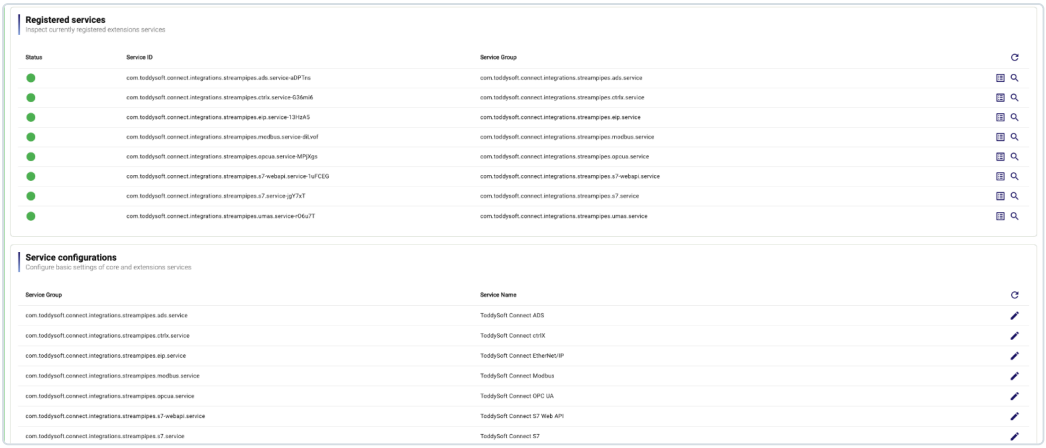


Adapters saved with the old version keep running after an update.

Licence

The driver reads its licence from the service's configuration in StreamPipes:

1. Open **Settings → Extension Services**. *Registered services* lists the running ToddySoft services; **Service configurations** below holds one entry per service.



2. Press the pencil next to *ToddySoft Connect OPC UA*. The entry has one field, `licence-opcua`.



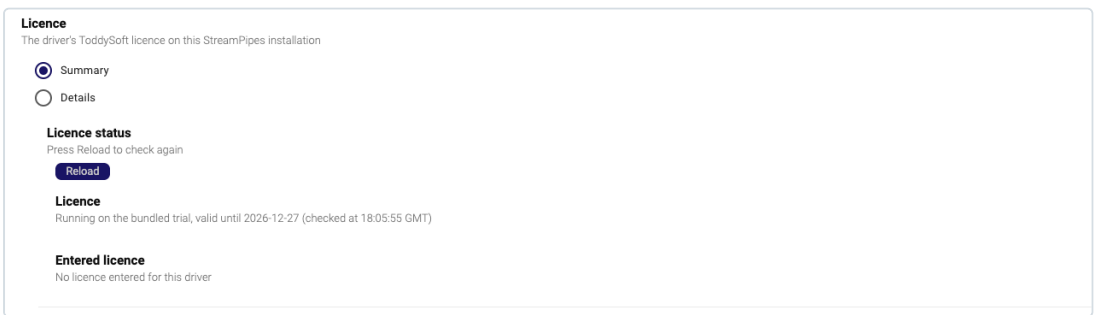
3. Paste the licence file **Base64-encoded**: `base64 -i opcua.lic` (macOS) or `base64 -w0 opcua.lic` (Linux) prints it.

4. **Click into the field once more after pasting**, then press **Update**. StreamPipes takes the field's value only on that click; saving straight after pasting stores the previous value, usually empty.

The licence takes effect on the next browse or adapter start; no restart is needed. Without a licence, or with one the driver rejects, the bundled trial applies until it expires. The value is masked in the form but stored in StreamPipes' database as entered, so limit who can administer StreamPipes.

Checking the licence

Every adapter form opens with a **Licence** block, folded to its **Summary**: whether the driver is licensed and by which licence, until when, and what became of the licence you entered. **Details** shows everything the driver reports about the licence in use: its checksum (which identifies it), whom it is licensed to, how long it is valid and until when support runs, and its tier. Press **Reload** in the block after changing the licence.



the block says	meaning	what to do
Licensed by the licence entered in the service configuration · Entered licence: In use	your licence applies	nothing
Running on the bundled trial · No licence entered for this driver	no licence entered; the trial applies until the date shown	enter the licence (above)
Entered licence: Not in use: the entered value cannot be read as a licence (position N)	the field holds something that is not a Base64-encoded licence file, usually extra characters copied along	paste the encoded file again, nothing before or after it
Entered licence: Not in use: refused by the driver	the licence is genuine but does not cover this: another driver, another platform, expired	check the licence; the reason is in the service's log (`blob rejected for driver`)
Licensed by a licence file	a licence file in the service's licence directory applies	nothing
Attention: Expires in N days / Support ends in N days	the licence, or the driver versions it covers, ends soon	renew in time
Attention: Expired on and running in its grace period until	the licence has ended; ToddySoft Connect still accepts it for 30 days	renew now: after that date the licence is refused

Fields the driver version does not report yet (what the licence covers, when it was issued, its description) are shown as *not reported by this driver version*. Nothing in the block shows the licence itself.

Reaching the devices

A service in a container sees only its container networks. Connecting, browsing and reading go out through the machine's routing and work as long as the machine has a route to the device. **The search** does not: broadcasts, multicast and PROFINET-DCP never leave the container's network, so on its own the search finds nothing on the plant network. There are two ways to give it the plant network.

The ToddySoft discovery agent (recommended)

The agent is a small container that runs once per machine with the host's network and searches the plant networks for every ToddySoft service that asks. It only searches: the adapter still connects from its own container. It comes as an archive next to this service's, `discovery-agent-<version>-<architecture>.tar.gz`; use the agent of the same version.

```
docker load < discovery-agent-1.3.0-amd64.tar.gz
docker run -d --name toddysoft-discovery-agent --restart unless-
        stopped \
  --network host --cap-add NET_RAW \
  toddysoft/discovery-agent:1.3.0
```

- **Nothing to set on the service.** At every search it looks for the agent at its container network's gateway, which on Docker is the machine itself. An agent started later is found on the next Reload. The search's status line says *via discovery agent* when the agent answered, *no discovery agent found · container networks only* when there is none, or why the agent refused.
- **Licence.** The service asks only for OPC UA devices and sends the licence its driver runs on over an encrypted connection; the agent checks it, uses it for that one search and keeps nothing.
- **Security.** The agent listens only on the machine's Docker-network addresses (TCP 47900) and accepts connections only from the container networks; allow that port from them if the machine has a firewall. For more, set a shared secret: `-e TODDYSOFT_AGENT_SECRET=<secret>` on the agent and `-e TODDYSOFT_DISCOVERY_AGENT_SECRET=<secret>` on the service.
- **Discovering is not reaching.** A device the agent finds on a network the machine cannot route to is listed and then fails to connect.

A second network on the plant interface

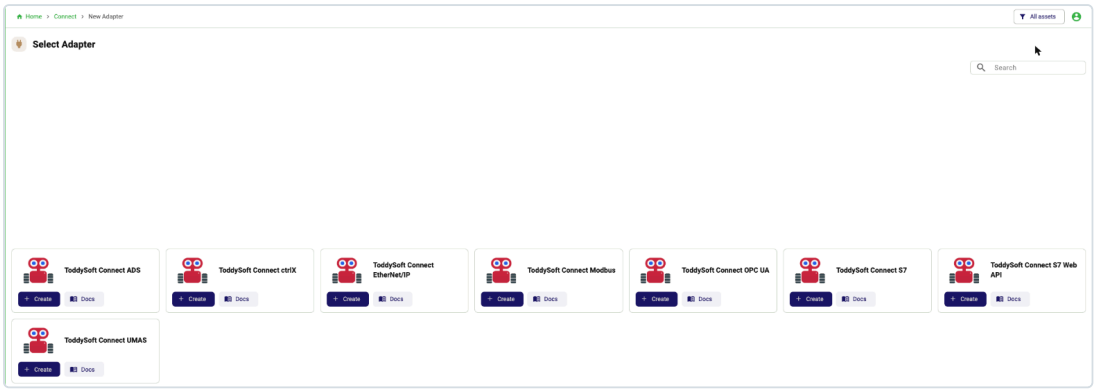
Alternatively, connect the service to a macvlan network on the plant interface, in the devices' own subnet, as a **second** network (it keeps `spnet`):

```
docker network create -d macvlan --subnet 192.168.23.0/24 --gateway
        192.168.23.1 -o parent=eth1 plant
docker network connect --ip 192.168.23.250 plant toddysoft-opcua
```

Each service needs this, once per plant subnet. Set `-e TODDYSOFT_DISCOVERY_AGENT=off` on a service whose only network is a macvlan, so it does not look for an agent at the plant router.

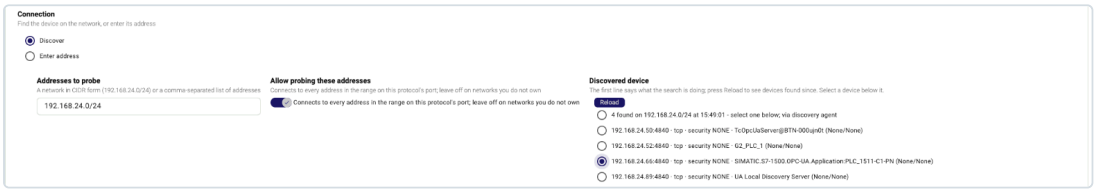
Creating an adapter

Open **Connect** → **New Adapter** and press **Create** on *ToddySoft Connect OPC UA*.



Finding the device

OPC UA devices do not announce themselves, so they are found by **trying addresses**: the service connects to every address of a range on the protocol's port. That is done only with your go-ahead.



1. Under **Connection** → **Discover**, enter the **Addresses to probe**: a network such as `192.168.24.0/24`, or a comma-separated list of addresses.
2. Switch on **Allow probing these addresses**. Leave it off on networks you do not own.
3. Press **Reload** next to **Discovered device**.

Nothing is probed before both are set. With the discovery agent the range and the consent are passed to the agent unchanged; consent with an empty range probes the agent's plant networks.

The first line of **Discovered device** says what the search is doing: it runs in the background for up to 25 seconds, and **Reload** shows the devices found since. Select a device below that line. If nothing is found, the line names the networks searched and any range that was refused. A finished search is reused for a minute.

Security

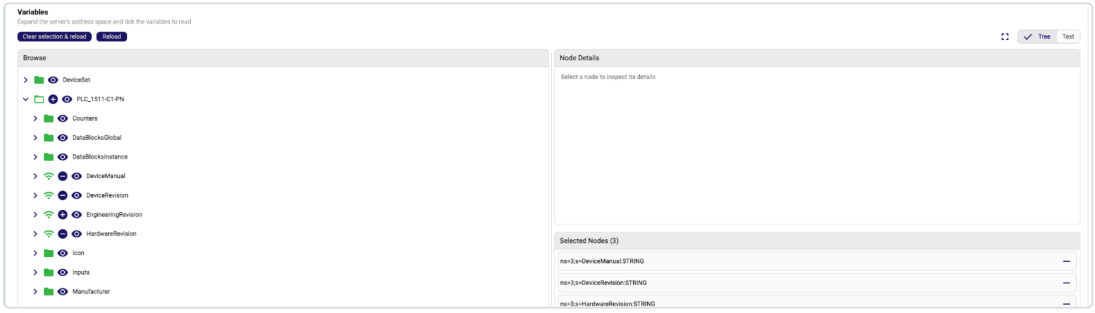
The connection is currently **unsecured and anonymous**: only endpoints this adapter can open without security are listed and used. Secured channels and user credentials are planned. Use it only on a network you trust, or against a server that offers an unsecured endpoint for reading.

Settings

Setting	Meaning
Connection	*Discover* a server, or *Enter address* to type the fields below yourself
Addresses to probe	Network (`192.168.24.0/24`) or comma-separated addresses; see above
Allow probing these addresses	Consent to connect to every address of the range
Host, Port, Endpoint path	Where the server is: `opc.tcp://host:4840/path` is host, port and path
Polling interval (ms)	How often the selected tags are read; every read produces one event
Add timestamp	Adds the field `timestamp` (the time of the read) to every event; see below
Tags	What is read; see below

The **Documentation** link at the top of the form opens a short version of this page in StreamPipes.

Choosing the tags



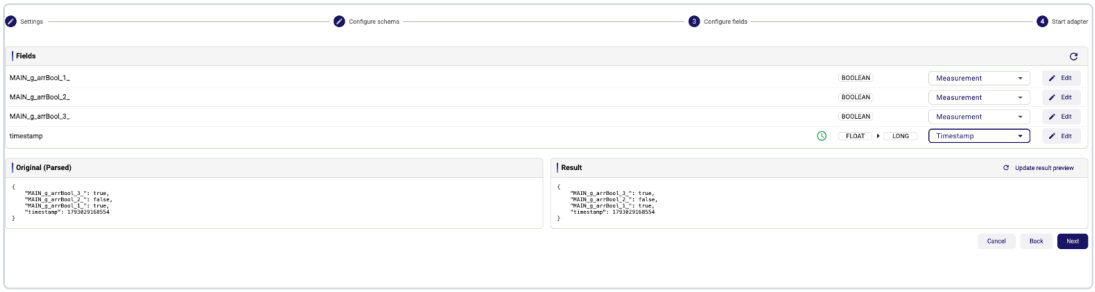
The server's address space is browsed **one level at a time** as you expand it. Expand folders and objects to find variables; only variables with a known data type can be selected. Each becomes one field, named after its node id (`ns=2;s=Demo.Dynamic.Double` becomes `Demo_Dynamic_Double`).

The tree's **Text** mode edits the selection directly, one address per line - useful to paste a list of addresses. StreamPipes puts its own comment lines into that editor (`# Provide OPC UA Node IDs …`); lines starting with `#` are ignored.

Events and the timestamp

Press **Next**. StreamPipes reads the tags once and shows the sample event; every poll then produces one event like it, with one field per tag that could be read and `timestamp`, the time of the read in milliseconds since the epoch.

StreamPipes cannot be told by the adapter which field is the timestamp. In **Configure fields**, set `timestamp`'s role to **Timestamp** and its type to **Long**:



- StreamPipes 0.99 types **every number as a float** in the schema it infers. The events themselves carry the exact values; change a field's type here where it matters.
- A tag that is itself named `timestamp` arrives as `timestamp_tag`.
- **Add timestamp** off leaves the field out, for a pipeline that sets its own.

Then name the adapter and start it under **Start adapter**.

Troubleshooting

- **The form looks like an older version, or a setting described here is missing**: update the installed adapter (Settings → Extensions → "…" → Update).
- **The tag tree shows an error**: the message names the cause - an unreachable address, a refused login, a missing licence. Where it says only *An unknown error has occurred*, the service's log has the details: `docker logs toddysoft-opcua`.
- **The search finds nothing**: read its status line. *Container networks only* means neither the discovery agent nor a plant network is there (see Reaching the devices).
- **The licence is not taken**: click into the field after pasting, then Update. A licence the driver rejects is logged as `blob rejected for driver`, and the trial applies; the adapter form's Licence block says which licence applies and why yours does not (see Checking the licence).
- **The service does not appear under Extensions**: check `docker logs toddysoft-opcua`; usually `SP_CLIENT_SECRET` or the network is wrong.
- **After changing only the endpoint path, the tree still shows the old server**: press **Reload** next to the tag tree; StreamPipes does not reload it for that field.

Version history

1.3.0 — 2026-11-13

Built on ToddySoft Connect 1.3.0.

- **New:** the first release of the OPC UA adapter for Apache StreamPipes 0.99 - device search, the tag tree, polling into StreamPipes events, the licence in the service configuration and its status in every adapter form, and search from containers through the ToddySoft discovery agent.