ToddySoft Connect ctrlX - Apache Kafka Connect

A Kafka Connect **source and sink connector** for Bosch Rexroth ctrlX CORE controllers, built on the ToddySoft Connect ctrlX driver. It reads nodes of the ctrlX Data Layer - PLC variables, axis state, system metrics - through the Data Layer's REST API over HTTPS, with a ctrlX user account, and publishes every read as one Kafka record; the sink writes record fields to nodes.

The connector runs on **your own Kafka Connect workers** near the plant: Apache Kafka's Connect or Confluent Platform, as a service or in a container. It does **not** run in Confluent Cloud: a managed or custom connector there runs in Confluent's network and cannot reach a controller on a plant network.

Compared with the Apache PLC4X connector

Apache PLC4X, the open-source project ToddySoft Connect's drivers descend from, publishes a free Kafka Connect connector. As of PLC4X 1.0.0 the differences are:

	this connector	**Apache PLC4X connector 1.0.0**
Credentials	handed to the driver as such, supplied by Kafka config providers; never in the connection URL, never logged	only what fits into the connection string
Connector configuration in the worker log	never logged	every property logged when a task starts, connection strings included
When tags are read	on a timer, on a value change (with a condition), or by subscription	at a fixed interval
Configuration	one versioned file, shared with the Camel component, checked before start with every problem located	flat connector properties
Several identical PLCs	one batch over a tag set for many connections, spread over tasks	one source per PLC
Packaging	one plugin per driver, as plugin zip and Confluent Hub component archive	one plugin with every PLC4X driver
Licence and support	commercial licence; support from ToddySoft	Apache License 2.0, free; community support

Requirements

	supported	tested
Apache Kafka (Connect)	3.9.x, 4.0, 4.1, 4.2, 4.3, 4.4	3.9.2, 4.4.0
Confluent Platform	7.9, 8.0, 8.1, 8.2, 8.3, 8.4	8.4
Java on the worker	17 or newer	21, 25

- **Kafka older than 3.9 and Java older than 17 are not supported.** (On Java 11 the worker cannot load the plugin at all; see Troubleshooting.)
- **Self-managed workers only**, standalone or distributed. Every worker that may run a task needs the plugin, the configuration file (if you use one) and the licence.
- **A route from each worker to the controllers** (TCP port 443, HTTPS).
- **On the controller**: the Data Layer service running, and a **user account** for the connector (see Credentials). Reading needs the permission *Data Layer Global → View resources* (`rexroth-automation.datalayer.r`); *Data Layer → View* is not it. ctrlX manages reading and writing as separate permissions: a read-only account suffices for the source connector, the sink needs an account allowed to write.
- A **ToddySoft licence for ctrlX** for production use. Without one the driver runs on its bundled trial licence until that expires.

Installation

You receive the plugin twice, in the share you received your licence and downloads from; both hold the same connector:

- `toddysoft-kafka-connect-ctrlx-<version>.zip` - Confluent Hub's component archive format, for Confluent's installer;
- `toddysoft-kafka-connect-ctrlx-<version>-plugin.zip` - a plain plugin directory.

With Confluent's CLI (Confluent Platform)

```
confluent connect plugin install ./toddysoft-kafka-connect-ctrlx-1.3.0.zip \
  --plugin-directory /usr/share/confluent-hub-components \
  --worker-configurations /etc/kafka/connect-distributed.properties --force
```

The CLI installs from the local file, without internet access, into `<plugin-directory>/toddysoft-kafka-connect-ctrlx` and adds the directory to `plugin.path` in the worker configuration. `--force` accepts the licence agreement it shows; leave it out to read and accept it yourself. It does not run on Windows, and not while logged in to Confluent Cloud. Older Confluent Platform releases have `confluent-hub install <zip> --component-dir <dir> --worker-configs <file> --no-prompt` instead.

By unzipping (any worker)

```
sudo unzip toddysoft-kafka-connect-ctrlx-1.3.0-plugin.zip -d
        /opt/kafka/plugins/
```

`/opt/kafka/plugins` must be listed in the worker's `plugin.path` (`plugin.path=/opt/kafka/plugins`). Restart every worker afterwards.

In a container

Mount the unzipped plugin and the licence directory into the worker container, for example with the official `apache/kafka` image (it runs as `appuser`, home `/home/appuser`):

```
docker run -d --name connect \
  -v /opt/kafka/plugins:/opt/kafka/plugins:ro \
  -v /etc/toddysoft/licenses:/home/appuser/.toddysoft/licenses:ro \
  -v /etc/kafka-connect:/etc/kafka-connect:ro \
  apache/kafka:4.4.0 /opt/kafka/bin/connect-distributed.sh /etc/kafka-
        connect/worker.properties
```

`confluentinc/cp-kafka-connect` runs as `appuser` with the same home; its default plugin path is `/usr/share/confluent-hub-components`.

Checking the installation

```
curl -s http://localhost:8083/connector-plugins | grep -i ctrlx
```

lists `com.toddysoft.connect.integrations.kafka.ctrlx.CtrlxSourceConnector` and `…CtrlxSinkConnector` with their version.

Updating

Remove the old plugin directory before installing the new one (the Confluent CLI does this itself): with two versions of one plugin on `plugin.path`, the worker loads either. Restart every worker.

Licence

The driver reads its licence from **`.lic` files in `~/.toddysoft/licenses/` of the user the worker runs as**:

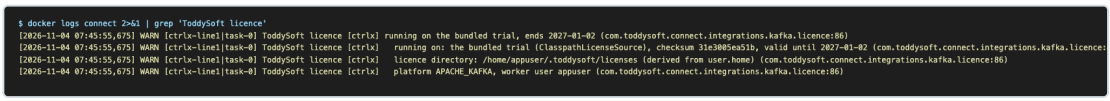
the worker runs	put the licence file in
as a service, user `kafka`	`~kafka/.toddysoft/licenses/ctrlx.lic` (for example `/home/kafka/.toddysoft/licenses/`)
in the `apache/kafka` or `cp-kafka-connect` container	a host directory mounted at `/home/appuser/.toddysoft/licenses` (see above)
in a distributed cluster	the same on **every** worker: a task moved to a worker without it runs on the trial there

The file name does not matter, only the extension `.lic`; one file per driver or one licence covering several drivers both work. A new licence counts from the next connector or task start (restart the connector). Without a licence, or with one the driver rejects, the bundled trial applies until it expires; the driver then refuses its connections.

Checking the licence

Each task writes a **licence report** to the worker log when it starts, one block per driver, every line beginning `ToddySoft licence [ctrlx]`:

```
ToddySoft licence [ctrlx] running on the bundled trial, ends 2027-01-02
ToddySoft licence [ctrlx]   running on: the bundled trial
(ClasspathLicenseSource), checksum 31e3005ea51b, valid until 2027-01-02
ToddySoft licence [ctrlx]   found and rejected: a licence file
(FileSystemLicenseSource), checksum 9be41f07c2aa: PLATFORM_NOT_LICENSED:
License does not entitle this platform
ToddySoft licence [ctrlx]   licence directory:
/home/appuser/.toddysoft/licenses (derived from user.home)
ToddySoft licence [ctrlx]   platform APACHE_KAFKA, worker user appuser
```



Find it with `grep 'ToddySoft licence' <worker log>` (in a container: `docker logs connect 2>&1 | grep 'ToddySoft licence'`). A licence is named by its **checksum**, the first 12 characters of its SHA-256: `shasum -a 256 ctrlx.lic` shows which file is meant.

the report says	meaning	what to do
`licensed` · running on: a licence file	your licence applies	nothing
`licensed, expires in N days`	it ends soon	renew in time
`licensed in its grace period: expired on <date>, refused after <date>`	it has ended; ToddySoft Connect still accepts it for 30 days	renew now: after that date the licence is refused
`attention: Support ends in N days (<date>)` / `attention: Support ended on <date>`	driver versions built after that date will not run on this licence	renew the support before updating the plugin
`running on the bundled trial, ends <date>` and no *found and rejected* line	no licence file found	put the file in the directory the report names, for the user it names
… and `found and rejected: …: PLATFORM_NOT_LICENSED`	the licence does not cover Kafka Connect	ask ToddySoft for a licence that does
… `DRIVER_NOT_LICENSED`	it is a licence for another driver	use the ctrlX licence
… `LICENSE_EXPIRED`	it has expired	renew
`NOT licensed: …`	no licence applies, not even the trial; the driver refuses every connection	as above

A licence refusal on a connection is reported once with the same block, then counted (`refused N connections since …, licence unchanged`, at most every ten minutes) instead of filling the log. Nothing in the log shows the licence itself or whom it is licensed to. The task does not fail on a licence: the connector stays RUNNING and its records carry the refusal in `errors`; the report is where the cause is.

Configuration file

Both connectors run a ToddySoft Connect **event-pump configuration** (version 2): the connections with their credentials, the tags by alias, and batches - which tags are read together, and when. The archive's `etc/` holds a sample; a complete one:

```
{
  "version": 2,
  "connections": [
    {
      "id": "line1",
      "url": "ctrlx://192.168.10.20:443",
      "credential": { "method": "username-password",
                      "username": "{{line1.username}}", "password": "
        {{line1.password}}" },
      "tags": {
        "running":  "plc/app/Application/sym/PLC_PRG/running",
        "speed":    "plc/app/Application/sym/PLC_PRG/speed",
        "counter":  "plc/app/Application/sym/PLC_PRG/counter",
        "setpoint": { "address": "plc/app/Application/sym/PLC_PRG/setpoint",
        "access": "write" }
      }
    }
  ],
  "batches": [
    { "id": "process", "connectionId": "line1", "tags": ["running", "speed",
        "counter"],
      "trigger": { "type": "timer", "intervalMillis": 1000 } },
    { "id": "state", "connectionId": "line1", "tags": ["running",
        "counter"],
      "trigger": { "type": "change", "watch": ["running"],
        "pollIntervalMillis": 200 } }
  ]
}
```

- **Connection URL**: `ctrlx://<host>:443` (443 is the default and may be left out). Further parameters are appended as a query: `request-timeout-ms` (one HTTPS request, default 5000), `base-path` (the Data Layer REST API, default `/automation/api/v2`), and the certificate parameters below: `ctrlx://192.168.10.20:443?request-timeout-ms=10000`. The account never goes into the URL.
- **Addresses** are Data Layer node paths, slash-separated as ctrlX shows them: a PLC variable is `plc/app/<application>/sym/<program>/<variable>`, for example `plc/app/Application/sym/PLC_PRG/counter`; a member of a structure `plc/app/Application/sym/PLC_PRG/g_simple/dat`; an axis's actual position `motion/axs/<axis>/state/values/actual/pos`; a system metric

`framework/metrics/system/cpu-utilisation-percent`. A leading `/` is optional. A pattern such as `framework/metrics/system/**` is for browsing, not a tag address.
- **Triggers**: `timer` (every `intervalMillis`), `change` (read the batch when a watched tag changes, optionally under a `condition` such as `oldValue == false && value == true`), `subscription` (the driver subscribes and forwards each change; the controller publishes changes at most every `publish-interval-ms`, a URL parameter, default 1000).
- **Access**: `read` tags are never written, `write` tags are never read; without `access` a tag is both.
- **Several identical PLCs**: see below.

Give the file to the connector either inline (`configuration`) or as a path on **every** worker (`configuration.file`). A configuration with a problem refuses the connector, listing each problem with its location in the file and a rule id.

The controller's certificate

A ctrlX CORE answers HTTPS with a **self-signed certificate** that chains to no public CA and names the device rather than its address. The driver verifies it (`tls.verify`, on by default - keep it on; it checks the hostname too) and refuses a certificate it cannot verify: the worker log names its SHA-256 fingerprint (`SHA-256 fingerprint: a3:f8:…:26:34`), and once the connection has failed the records carry the reason in `errors`. Check that fingerprint against the controller (its web interface shows the certificate) and pin it in the URL:

```
ctrlx://192.168.10.20:443?tls.pinned-fingerprints=a3:f8:…:26:34
```

A pin accepts exactly the certificates listed and overrides `tls.verify`; the hostname is not checked then, since the pin already identifies the controller. **A ctrlX CORE may present more than one certificate**, one per connection at random: pin every one, comma-separated. A connection that draws one not yet pinned is refused with `TLS pin check failed: peer certificate SHA-256 fingerprint <value> is not in the pinned list` - add that value to the list rather than replacing it.

A certificate signed by a CA that the worker's Java trusts needs no pin, as long as it names the address in the URL; the ctrlX driver takes no trust-store parameter of its own.

Credentials

A credential never goes into the URL or the file itself. Every secret field is a **reference** `{{name}}`, and the connector property `secrets.<name>` supplies the value - best through a Kafka **config provider**, so the value stays out of the connector configuration too. Enable the file provider on every worker:

```
config.providers=file
config.providers.file.class=org.apache.kafka.common.config.provider.FileConfig
```

and refer to a properties file readable only by the worker's user:

```
"secrets.line1.username": "${file:/etc/kafka-
        connect/secrets/plc.properties:line1.username}",
"secrets.line1.password": "${file:/etc/kafka-
        connect/secrets/plc.properties:line1.password}"
```

the controller	`method`	fill in
ctrlX CORE (every Data Layer request needs a login)	`username-password`	`username`, `password`

The username is **required**: the Data Layer has no anonymous mode, and a connection without a credential is refused. The password must be a reference; the username may also be written into the file as it is. Use an account of its own for the connector, with no more rights than it needs (see Requirements).

Kafka never logs these connectors' configuration values, and the worker writes the `${file:…}` placeholder, not the value, into its config topic. Restrict the config topic with ACLs all the same.

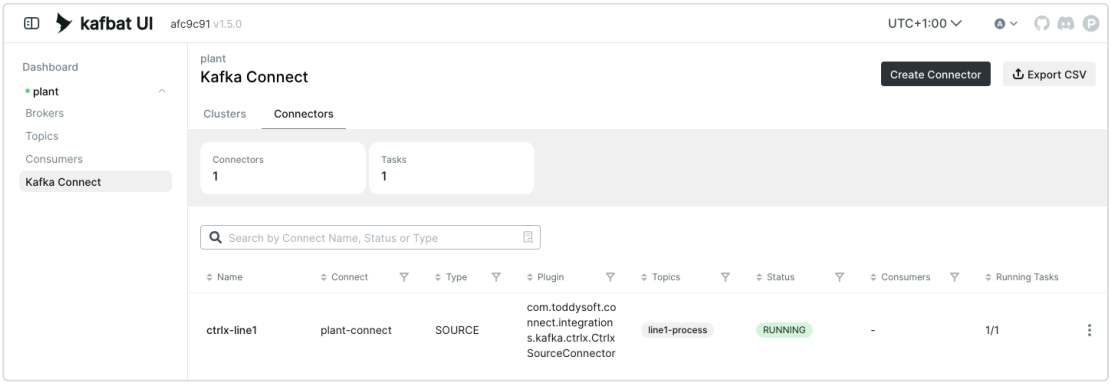
Source connector

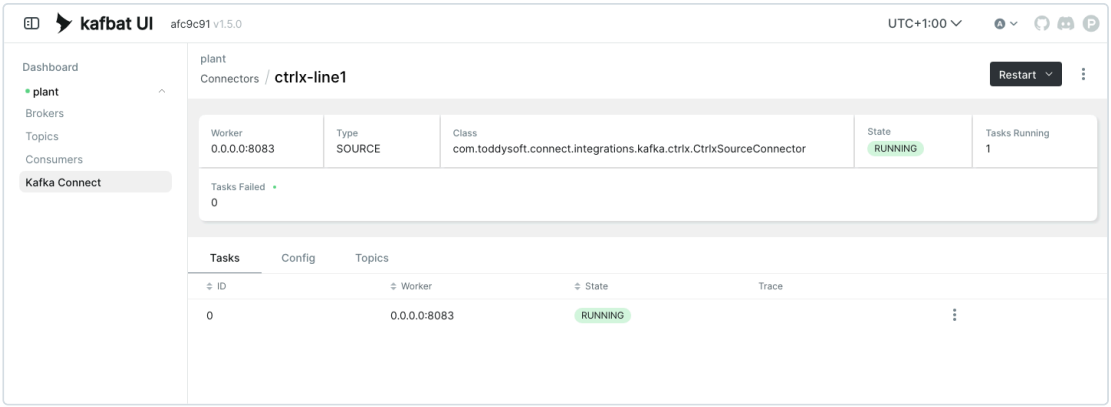
```
{
  "name": "ctrlx-line1",
  "config": {
    "connector.class":
        "com.toddysoft.connect.integrations.kafka.ctrlx.CtrlxSourceConnector",
    "configuration.file": "/etc/kafka-connect/line1.toddysoft.json",
    "batches.topics": "{\"process\": \"line1-process\", \"state\": \"line1-
        state\"}",
    "topic.default": "line1-other",
    "secrets.line1.username": "${file:/etc/kafka-
        connect/secrets/plc.properties:line1.username}",
    "secrets.line1.password": "${file:/etc/kafka-
        connect/secrets/plc.properties:line1.password}",
    "tasks.max": "1"
  }
}
```

`curl -X POST -H 'Content-Type: application/json' --data @ctrlx-line1.json http://localhost:8083/connectors` creates it.

property	default	meaning
`configuration` / `configuration.file`	-	the configuration, inline or as a path on every worker; exactly one
`batches`	all	the batch ids to run, comma-separated
`batches.topics`	`{}`	JSON object: batch id to topic
`topic.default`	-	topic of every batch not in `batches.topics`
`secrets.<name>`	-	the value of `{{name}}`

property	default	meaning
`tasks.max`	1	batches (and connections, see below) are spread over at most this many tasks





The screenshots show kafka-ui (kafbat UI), a free web interface for Kafka and Kafka Connect; any tool that shows connectors and topics does, Confluent Control Center on Confluent Platform among them. They were taken against a ctrlX CORE reading `plc/app/Application/sym/PLC_PRG/g_b1`, `g_s16` and `g_s32` once a second.

Records

One record per read. Its value is a `Struct`:

field	type	
`batch`	STRING	the batch id
`connection`	STRING	the connection id
`timestamp`	INT64	the time of the read, milliseconds since the epoch
one field per tag alias	see the type table	`null` when the tag could not be read
`errors`	MAP<STRING,STRING>	alias to the reason, for every tag that could not be read

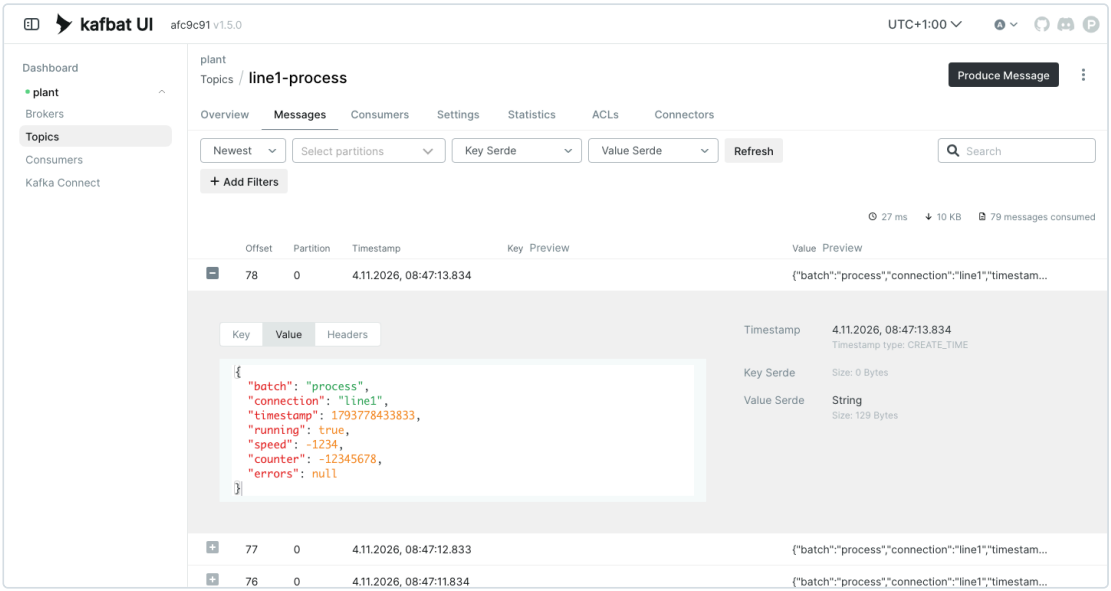
The record has **no key**; its source partition is `{batch, connection}`, its offset `{timestamp}`.

Each Data Layer node reports its own type; the connector maps it:

Data Layer type	Connect type	
bool8	BOOLEAN	
int8	INT8	
uint8, int16	INT16	unsigned 8-bit widened
uint16, int32	INT32	unsigned 16-bit widened
uint32, int64	INT64	unsigned 32-bit widened
uint64	Decimal (scale 0)	unsigned 64-bit, exact
float, float32	FLOAT32	
double, float64	FLOAT64	
string	STRING	
timestamp	Timestamp	
object (a structure)	STRING	JSON
an array (arbool8, arint16, …)	ARRAY of STRING	each element as text

The Data Layer has **no time types** of its own: a PLC's `TIME` and `TIME_OF_DAY` are uint32 nodes and arrive as the count they hold (milliseconds), an `LTIME` is a uint64 node and arrives in nanoseconds; `DATE` and `DATE_AND_TIME` are timestamp nodes. Inside a structure, dates and durations arrive as the plain integer the controller holds; read such a member on its own address to get it typed.

A field takes its type from the batch's **first read**, and every field is optional: a tag that could not be read on the first read is a STRING from then on - restart the connector once the controller answers.



Converters and data formats

The connector produces Connect `Struct` records with a schema; the worker's converter decides the bytes on the topic:

- `org.apache.kafka.connect.json.JsonConverter` with `value.converter.schemas.enable=true` (schema and payload) or `false` (payload only);
- `io.confluent.connect.avro.AvroConverter` or `io.confluent.connect.protobuf.ProtobufConverter` with Confluent's Schema Registry (`value.converter.schema.registry.url=http://registry:8081`), on a worker that has those converters (Confluent Platform has them).

Set the converter per connector (`"value.converter": "…"`) or for the worker. Every read of a batch has the same schema; a tag added to the batch changes it.

Sink connector

Writes the fields of each record to the tags of one connection, by alias:

```
{
  "name": "ctrlx-line1-setpoints",
  "config": {
    "connector.class":
        "com.toddysoft.connect.integrations.kafka.ctrlx.CtrlxSinkConnector",
    "configuration.file": "/etc/kafka-connect/line1.toddysoft.json",
    "connection": "line1",
    "topics": "line1-setpoints",
    "secrets.line1.username": "${file:/etc/kafka-
        connect/secrets/plc.properties:line1.username}",
    "secrets.line1.password": "${file:/etc/kafka-
        connect/secrets/plc.properties:line1.password}"
  }
}
```

property	default	meaning
`connection`	-	the connection id written to
`topics`	-	the topics read
`max.retries`	3	how often a write the controller failed is tried again
`retry.backoff.ms`	1000	the first wait before a retry, doubled each time
`retry.backoff.max.ms`	30000	the longest wait

A record is a `Struct` (with a converter that carries a schema, for example JSON with schemas or Avro); each field names a tag alias of the connection, and its type follows the type table. The driver writes a number in the node's own type, and refuses one the node cannot hold, or a fraction for an integer node, before anything is sent. A field the connection does not define, or a tag whose access is `read`, is refused: the record goes to Kafka Connect's error reporting (a dead letter queue with `errors.tolerance=all` and `errors.deadletterqueue.topic.name`)

unwritten. A write the controller fails is retried, then reported the same way. A write refused for a licence reason is not retried. The sink's account needs the right to write; an account that only reads is refused with HTTP `403`.

Several identical PLCs

A batch with `connectionIds` runs once per connection - usually reading a tag set every PLC has:

```
{
  "version": 2,
  "tagSets": { "press": { "speed": "plc/app/Application/sym/PLC_PRG/speed",
                          "count": "plc/app/Application/sym/PLC_PRG/count" }
        },
  "connections": [
    { "id": "press-1", "url": "ctrlx://192.168.10.31:443", "tagSets":
        ["press"],
      "credential": { "method": "username-password", "username": "
        {{press.username}}", "password": "{{press.password}}" } },
    { "id": "press-2", "url": "ctrlx://192.168.10.32:443", "tagSets":
        ["press"],
      "credential": { "method": "username-password", "username": "
        {{press.username}}", "password": "{{press.password}}" } },
    { "id": "press-3", "url": "ctrlx://192.168.10.33:443", "tagSets":
        ["press"],
      "credential": { "method": "username-password", "username": "
        {{press.username}}", "password": "{{press.password}}" } }
  ],
  "batches": [
    { "id": "line-speed", "connectionIds": ["press-1", "press-2", "press-
        3"], "tags": ["speed", "count"],
      "trigger": { "type": "timer", "intervalMillis": 500 } }
  ]
}
```

Every connection needs its credential (here one account for all three presses) and, with factory certificates, the pin of its own controller. Each press's read is a unit of its own, spread over `min(units, tasks.max)` tasks: with `tasks.max` 3 on a three-worker cluster each worker reads one press. The records keep the batch id `line-speed` and name the connection, so one topic holds every press, partitioned by press.

In **distributed mode** Kafka Connect moves tasks between workers on a rebalance; a moved task connects from its new worker and writes its licence report there. A stopped worker's tasks move only after up to five minutes (`scheduled.rebalance.max.delay.ms`): Kafka Connect waits that long for the worker to come back. Standalone mode runs everything on one worker.

Several driver plugins on one worker

Each driver is its own plugin, and they sit side by side on one worker (`/opt/kafka/plugins/toddysoft-kafka-connect-ctrlx`, `…-s7`, …). **Use each driver's own connector classes** - `…kafka.ctrlx.CtrlxSourceConnector` here: a worker loads a connector class from one plugin only. The generic `PlcSourceConnector` / `PlcSinkConnector`

exist in every plugin and are not meant to be used; a task that finds itself without the driver its URL names refuses to start and names the class to use. Within one plugin, a source and a sink to the same controller share one connection.

Troubleshooting

symptom	cause	what to do
the plugin is not in `GET /connector-plugins`	not on `plugin.path`, or the worker was not restarted	check `plugin.path` and the directory, restart every worker
the plugin is not listed, and the worker log shows `Failed to discover SourceConnector in …` caused by `UnsupportedClassVersionError … (class file version 61.0)`	the worker runs on Java 11 or older	run the worker on Java 17 or newer
a task fails with `driver-missing`	the generic or another driver's connector class	use `…kafka.ctrlx.CtrlxSourceConnector` / `CtrlxSinkConnector`
the connector is refused with a rule id and a location	the configuration has a problem	fix the file at each location named
RUNNING, but every record has the tags `null` and the reason in `errors`	the controller refuses or cannot be reached	read `errors`: an address, a refused login, a certificate, a licence
`errors` mentions `License validation failed`	no licence applies	see Checking the licence
the certificate is refused; the worker log names a `SHA-256 fingerprint`	the controller's self-signed certificate	pin the fingerprint (see The controller's certificate)
now and then `TLS pin check failed: peer certificate SHA-256 fingerprint <value> is not in the pinned list`	the controller presents several certificates, and not all are pinned	add that value to `tls.pinned-fingerprints`, comma-separated
`Driver 'ctrlx' requires a credential and none was supplied`	the connection has no `credential`	add one with `username-password` (see Credentials)

symptom	cause	what to do
HTTP `401 Unauthorized`	wrong username or password	check the account and the values `secrets.<name>` supplies
HTTP `403 Forbidden`	the account logged in but lacks the permission: reading needs *Data Layer Global → View resources*, writing a permission to write	adjust the account's role in the controller's user management
HTTP `404 Not Found` for a tag	the node path does not exist	check the path; PLC variables sit under `plc/app/<application>/sym/<program>/`
`INVALID_ADDRESS` for a tag	whitespace or an empty segment in the address	correct the address; the other tags of the batch are read
odd behaviour after an update	two copies of the plugin on `plugin.path`: the worker lists it once and logs nothing, so you cannot tell which copy runs	remove the old directory, restart

Raise the driver's log level for more detail: `curl -X PUT -H 'Content-Type: application/json' --data '{"level":"DEBUG"}' http://localhost:8083/admin/loggers/com.toddysoft` (and back to `INFO` afterwards).

Version history

1.3.0 — 2026-11-13

Built on ToddySoft Connect 1.3.0.

- **New:** the ctrlX connector as a product for self-managed Kafka Connect workers - this manual, a Confluent Hub component archive next to the plugin zip, both checked by every release; a licence report in the worker log per driver; supported versions Apache Kafka 3.9.x and 4.0-4.4, Confluent Platform 7.9 and 8.0-8.4.