ToddySoft Connect Modbus - Apache Kafka Connect

A Kafka Connect **source and sink connector** for Modbus TCP devices, built on the ToddySoft Connect Modbus driver. It reads coils, discrete inputs, input registers and holding registers from any Modbus TCP device - PLCs, meters, drives, I/O blocks and gateways - and publishes every read as one Kafka record; the sink writes record fields to coils and holding registers.

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 device 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 devices** (TCP port 502).
- **On the device**: its Modbus TCP server enabled. From its documentation you need its **Modbus map** - which register holds what, in which type and byte order - and its **unit identifier** if it is not 1: Modbus has no way to ask the device for either.
- A **ToddySoft licence for Modbus** 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-modbus-<version>.zip` - Confluent Hub's component archive format, for Confluent's installer;
- `toddysoft-kafka-connect-modbus-<version>-plugin.zip` - a plain plugin directory.

With Confluent's CLI (Confluent Platform)

```
confluent connect plugin install ./toddysoft-kafka-connect-modbus-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-modbus` 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-modbus-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 modbus
```

lists `com.toddysoft.connect.integrations.kafka.modbus.ModbusSourceConnector` and `…ModbusSinkConnector` 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**:

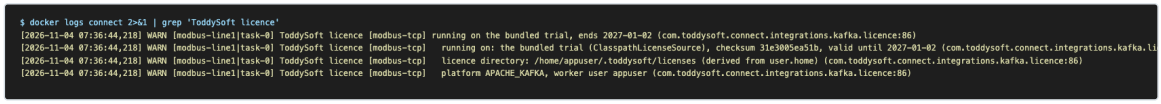
the worker runs	put the licence file in
as a service, user `kafka`	`~kafka/.toddysoft/licenses/modbus.lic` (for example `/home/kafka/.toddysoft/licenses/`)
in the `apache/kafka` or `cp-kafka-connect` container	a host directory mounted at `/home/appuser/.toddysoft/licenses` (see above)
in a distributed 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 [modbus-tcp]` (`modbus-tcp` is the Modbus TCP driver's code):

```
ToddySoft licence [modbus-tcp] running on the bundled trial, ends 2027-01-02
ToddySoft licence [modbus-tcp]   running on: the bundled trial (ClasspathLicenseSource), checksum
31e3005ea51b, valid until 2027-01-02
ToddySoft licence [modbus-tcp]   found and rejected: a licence file (FileSystemLicenseSource), checksum
9be41f07c2aa: PLATFORM_NOT_LICENSED: License does not entitle this platform
ToddySoft licence [modbus-tcp]   licence directory: /home/appuser/.toddysoft/licenses (derived from user.home)
ToddySoft licence [modbus-tcp]   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 modbus.lic` shows which file is meant. The report names the driver by its protocol code, `[modbus-tcp]`; the licence names it `MODBUS`, and a licence for another driver is refused with `License does not cover the 'MODBUS' driver`.

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 Modbus 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, 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": "modbus-tcp://192.168.10.20",
      "tags": {
        "running":     "holding-register:1:BOOL",
        "speed":       "holding-register:6:INT",
        "counter":     "holding-register:10:DINT",
        "temperature": "input-register:20:REAL",
        "setpoint":    { "address": "holding-register:30:INT", "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**: `modbus-tcp://<host>` (port 502; another port as `modbus-tcp://<host>:<port>`). The sample in `etc/` writes the transport out, `modbus-tcp:tcp://<host>:502`, which is the same. Parameters follow the `?`: `unit-identifier` (the unit / slave id, 0-255, default 1), `default-payload-byte-order` (see below), `request-timeout-ms` (one request as a whole, default 5000), `max-registers-per-request` (default 125), `max-coils-per-request` (default 2000), `max-address-gap-to-merge` (default 10), `max-concurrent-requests` (default 1): `modbus-tcp://192.168.10.20?unit-identifier=3&default-payload-byte-order=LITTLE_ENDIAN_BYTE_SWAP`.
- **Addresses** are `<area>:<register>[:<type>]`, the areas `holding-register`, `input-register`, `coil` and `discrete-input`: `holding-register:6:INT`, `holding-register:10:DINT` (registers 10-11), `input-register:20:REAL` (registers 20-21), `holding-register:40:STRING(20)` (a string of 20 characters; a `STRING` needs its length), `coil:3`, `discrete-input:7`. Without a type a register reads as `INT` and a coil or discrete input as `BOOL`. Register numbers **count from 1**: `holding-register:1` is the first holding register, offset 0 on the wire; add one to a device map that lists offsets from 0. An array selection goes before the type: `holding-register:100[0..9]:DINT` (ten DINTs, registers 100-119), `coil:1[0..7]:BOOL` (eight coils); `holding-register:100[9]:DINT` is **one** DINT, the tenth. Schneider's `%MW0` / `%M0` form is accepted too.
- **Per-tag options** in braces override the connection for one tag: `holding-register:100:INT{unit-id: 5}`, `holding-register:100[0..9]:DINT{unit-id: 5, byte-order: LITTLE_ENDIAN}`. `unit-id` reaches several units behind one connection, typically a serial line behind a Modbus gateway.
- **Byte order**: Modbus defines no layout for a value wider than one register, so devices differ. For a 32-bit value `AA BB CC DD`: `BIG_ENDIAN` (default, ABCD), `BIG_ENDIAN_BYTE_SWAP` (BADC), `LITTLE_ENDIAN_BYTE_SWAP` (CDAB, "word swap", low word first), `LITTLE_ENDIAN` (DCBA). A wrong setting does not fail, it returns plausible nonsense.
- **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; Modbus has no push, so the driver polls for it).
- **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.

Credentials

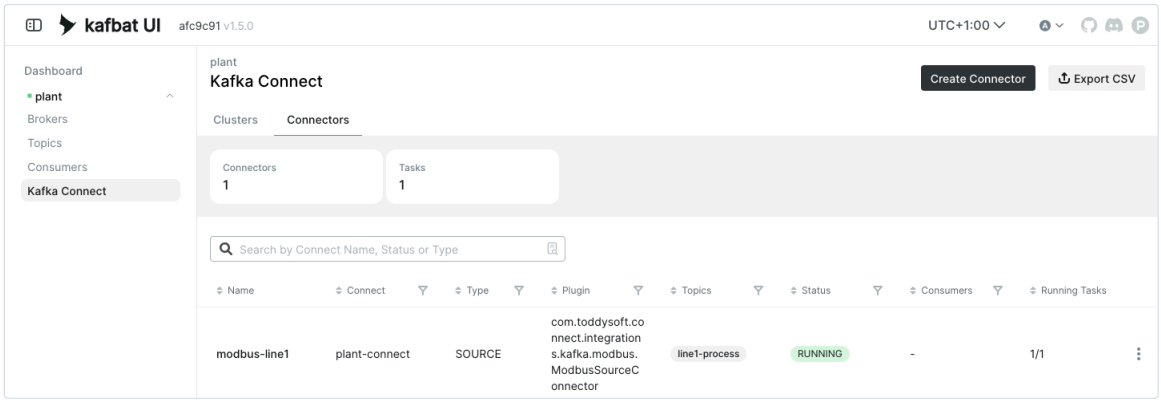
Modbus has no authentication: a Modbus connection has no `credential`, and the connector needs no `secrets.<name>` properties. Any host that reaches a device's port 502 can read and write it, so keep the devices in a network only the workers and other authorised clients reach.

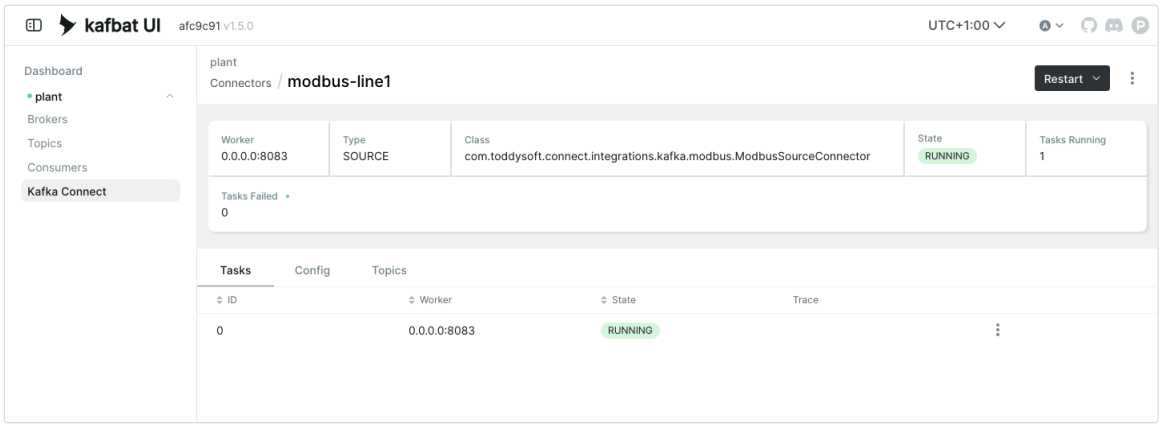
Source connector

```
{
  "name": "modbus-line1",
  "config": {
    "connector.class": "com.toddysoft.connect.integrations.kafka.modbus.ModbusSourceConnector",
    "configuration.file": "/etc/kafka-connect/line1.toddysoft.json",
    "batches.topics": "{\"process\": \"line1-process\", \"state\": \"line1-state\"}",
    "topic.default": "line1-other",
    "tasks.max": "1"
  }
}
```

`curl -X POST -H 'Content-Type: application/json' --data @modbus-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`
`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 the Modbus TCP server of an S7-1200 reading three holding-register addresses (BOOL, INT, DINT) once a second.

Records

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

field	type	
`batch`	STRING	the batch id

field	type	
`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

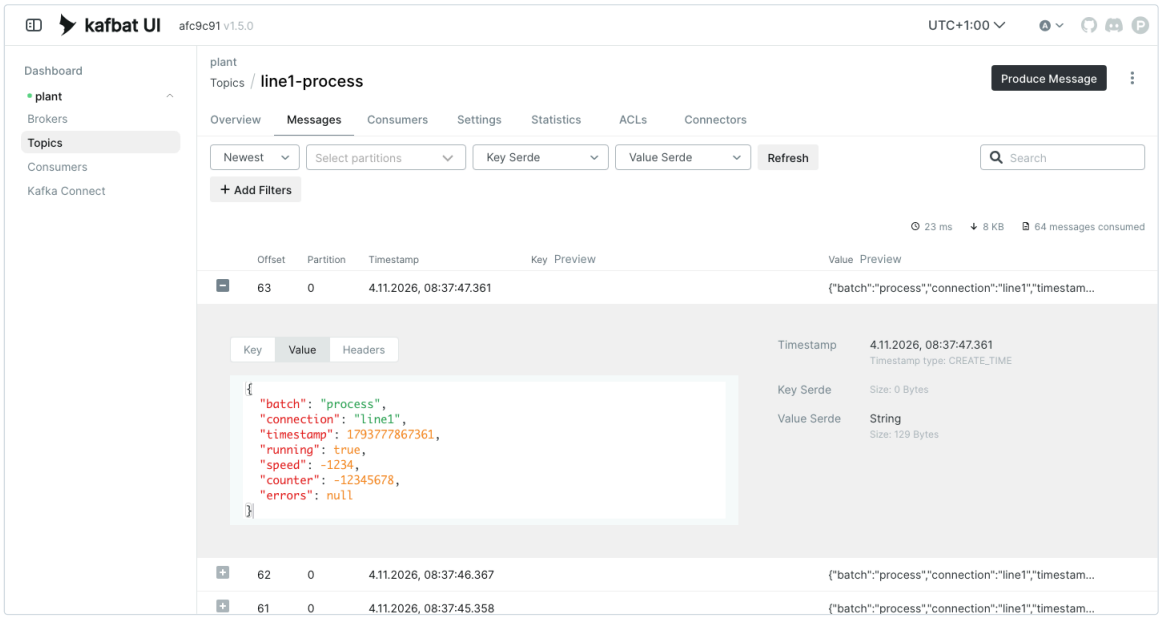
The record has **no key**; its source partition is `{batch, connection}`, its offset `{timestamp}`. With the configuration above a record of the batch `process` reads, as JSON without schema:


```
{"batch":"process","connection":"line1","timestamp":1793777835300,"running":true,"speed":-1234,"counter":-12345678,"err
```


type in the address	Connect type	
BOOL	BOOLEAN	
SINT	INT8	
BYTE, USINT, INT	INT16	unsigned 8-bit types widened
WORD, UINT, DINT	INT32	unsigned 16-bit types widened
DWORD, UDINT, LINT	INT64	unsigned 32-bit types widened
LWORD, ULINT	Decimal (scale 0)	unsigned 64-bit, exact
REAL	FLOAT32	
LREAL	FLOAT64	
CHAR, WCHAR, STRING, WSTRING	STRING	
TIME, LTIME	INT64	milliseconds
TIME_OF_DAY, LTIME_OF_DAY	INT64	milliseconds since midnight
DATE, LDATE	INT32	days since the epoch
DATE_AND_TIME, LDATE_AND_TIME	Timestamp	
an array	ARRAY of STRING	each element as text

Modbus defines no encoding for date and time; the driver reads them as unsigned counts - `TIME` in milliseconds, `DATE` and `DATE_AND_TIME` in seconds since the epoch, `TIME_OF_DAY` in milliseconds since midnight, the `L` variants the same in nanoseconds. A device that encodes them otherwise is read as an integer type and converted downstream.

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 device 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": "modbus-line1-setpoints",
  "config": {
    "connector.class": "com.toddysoft.connect.integrations.kafka.modbus.ModbusSinkConnector",
    "configuration.file": "/etc/kafka-connect/line1.toddysoft.json",
    "connection": "line1",
    "topics": "line1-setpoints"
  }
}
```

property	default	meaning
`connection`	-	the connection id written to
`topics`	-	the topics read
`max.retries`	3	how often a write the device 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. Modbus writes only **coils and holding registers**; input registers and discrete inputs are read-only by the protocol, so give their tags `"access": "read"`. 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 device fails is retried, then reported the same way. A write refused for a licence reason is not retried.

Several identical PLCs

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

```
{
  "version": 2,
  "tagSets": { "press": { "speed": "holding-register:6:INT", "count": "holding-register:10:DINT" } },
  "connections": [
    { "id": "press-1", "url": "modbus-tcp://192.168.10.31", "tagSets": ["press"] },
    { "id": "press-2", "url": "modbus-tcp://192.168.10.32", "tagSets": ["press"] },
    { "id": "press-3", "url": "modbus-tcp://192.168.10.33", "tagSets": ["press"] }
  ],
  "batches": [
    { "id": "line-speed", "connectionIds": ["press-1", "press-2", "press-3"], "tags": ["speed", "count"],
      "trigger": { "type": "timer", "intervalMillis": 500 } }
  ]
}
```

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-modbus` , `…-s7` , …). **Use each driver's own connector classes** - `…kafka.modbus.ModbusSourceConnector` 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 device 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.modbus.ModbusSourceConnector` / `ModbusSinkConnector`
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 device refuses or cannot be reached	read `errors` : a Modbus response code, a licence
`errors` mentions `License validation failed`	no licence applies	see Checking the licence
`Connection refused` on port 502 in the worker log	the device's Modbus TCP server is off, another port, or a firewall	enable Modbus TCP on the device, check the port and firewalls
a tag's reason in `errors` is `INVALID_ADDRESS`	the device answered *illegal data address* (exception 2): the register is outside its map, or the array runs past its end. Tags merged into one request share the answer, so a correct tag can report it for a neighbour	check every address and array length of the batch against the device's map; lower `max-address-gap-to-merge`
the values come from the neighbouring register	register numbers count from 1, the device map from 0	add one to the map's offsets
`UNSUPPORTED`	the device answered *illegal function* (exception 1): it does not serve that area or that write	use an area the device serves, per its map
`NOT_FOUND` , or no data from a device behind a Modbus gateway	the gateway reached no device at that unit (exception 11): the unit identifier is wrong	set `unit-identifier` in the URL, or `{unit-id: N}` on the tag
`REMOTE_ERROR` , and the worker log warns that a `response from unit N … was rejected`	an answer came from another unit than the one asked	check the gateway's unit mapping; for a device that does not echo the unit, `verify-unit-identifier=false`
plausible but wrong values: absurd floats, integers with their halves swapped	the device uses another byte order	set `default-payload-byte-order` in the URL, or `{byte-order: …}` on the tag (see Configuration file)
`REMOTE_BUSY` , and the worker log warns of no answer within `request-timeout-ms`	the request did not finish in time	raise `request-timeout-ms` , or put fewer tags in the batch
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). Every Modbus refusal is also logged at WARN with the exception's name and what it means.

Version history

1.3.0 — 2026-11-13

Built on ToddySoft Connect 1.3.0.

- **Fixed:** writes to extended registers (`6x`) work; every one failed before it was sent. Large merged writes stay within Modbus's limit per request.
- **New:** the Modbus 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.