ToddySoft Connect Modbus - Apache Kafka Connect

Ein Kafka-Connect-**Source- und Sink-Connector** für Modbus-TCP-Geräte auf Basis des Modbus-Treibers von ToddySoft Connect. Er liest Coils, Discrete Inputs, Input-Register und Holding-Register aus beliebigen Modbus-TCP-Geräten - Steuerungen, Zählern, Antrieben, E/A-Modulen und Gateways - und veröffentlicht jeden Lesevorgang als einen Kafka-Datensatz; der Sink schreibt Felder von Datensätzen in Coils und Holding-Register.

Der Connector läuft auf **Ihren eigenen Kafka-Connect-Workern** in der Nähe der Anlage: Connect aus Apache Kafka oder Confluent Platform, als Dienst oder im Container. Er läuft **nicht** in Confluent Cloud: Ein verwalteter oder eigener Connector läuft dort im Netz von Confluent und erreicht kein Gerät in einem Anlagennetz.

Im Vergleich zum Connector von Apache PLC4X

Apache PLC4X, das Open-Source-Projekt, aus dem die Treiber von ToddySoft Connect hervorgegangen sind, veröffentlicht einen freien Kafka-Connect-Connector. Stand PLC4X 1.0.0 sind die Unterschiede:

	dieser Connector	**Connector von Apache PLC4X 1.0.0**
Zugangsdaten	als solche an den Treiber übergeben, geliefert von Config Providern von Kafka; nie in der Verbindungs-URL, nie im Log	nur, was in die Verbindungszeichenfolge passt
Connector-Konfiguration im Log des Workers	wird nie protokolliert	jede Eigenschaft wird beim Start einer Task protokolliert, Verbindungszeichenfolgen eingeschlossen
Wann Variablen gelesen werden	per Timer, bei einer Wertänderung (mit Bedingung) oder per Abonnement	in einem festen Intervall
Konfiguration	eine versionierte Datei, gemeinsam mit der Camel-Komponente, vor dem Start geprüft, jeder Fehler mit seiner Stelle	flache Connector-Eigenschaften
Mehrere gleiche Steuerungen	ein Batch über einen Variablensatz für viele Verbindungen, verteilt auf Tasks	eine Quelle je Steuerung
Paketierung	ein Plugin je Treiber, als Plugin-ZIP und Komponentenarchiv für Confluent Hub	ein Plugin mit allen PLC4X-Treibern
Lizenz und Support	kommerzielle Lizenz; Support durch ToddySoft	Apache License 2.0, frei; Support durch die Community

Voraussetzungen

	unterstützt	**getestet**
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 auf dem Worker	17 oder neuer	21, 25

- **Kafka vor 3.9 und Java vor 17 werden nicht unterstützt.** (Unter Java 11 kann der Worker das Plugin gar nicht laden; siehe Fehlersuche.)
- **Nur selbst betriebene Worker**, standalone oder verteilt. Jeder Worker, der eine Task ausführen kann, braucht das Plugin, die Konfigurationsdatei (falls Sie eine verwenden) und die Lizenz.
- **Eine Route von jedem Worker zu den Geräten** (TCP-Port 502).
- **Im Gerät**: sein Modbus-TCP-Server eingeschaltet. Aus seiner Dokumentation brauchen Sie seine **Modbus-Belegung** - welches Register was enthält, in welchem Typ und welcher Byte-Reihenfolge - und seinen **Unit Identifier**, falls er nicht 1 ist: Modbus bietet keine Möglichkeit, das Gerät danach zu fragen.
- Eine **ToddySoft-Lizenz für Modbus** für den produktiven Einsatz. Ohne sie arbeitet der Treiber mit seiner mitgelieferten Testlizenz, bis diese abläuft.

Installation

Sie erhalten das Plugin in zwei Formen in der Freigabe, aus der Sie auch Lizenz und Downloads beziehen; beide enthalten denselben Connector:

- `toddysoft-kafka-connect-modbus-<version>.zip` - im Komponentenarchiv-Format von Confluent Hub, für Confluents Installationsprogramm;
- `toddysoft-kafka-connect-modbus-<version>-plugin.zip` - ein einfaches Plugin-Verzeichnis.

Mit dem Confluent 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
```

Das CLI installiert aus der lokalen Datei, ohne Internetzugang, nach `<plugin-directory>/toddysoft-kafka-connect-modbus` und trägt das Verzeichnis in `plugin.path` der Worker-Konfiguration ein. `--force` akzeptiert die angezeigte Lizenzvereinbarung; lassen Sie es weg, um sie selbst zu lesen und zu akzeptieren. Es läuft nicht unter Windows und nicht, solange Sie bei Confluent Cloud angemeldet sind. Ältere Versionen von Confluent Platform haben stattdessen `confluent-hub install <zip> --component-dir <dir> --worker-configs <datei> --no-prompt`.

Durch Entpacken (jeder Worker)

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

`/opt/kafka/plugins` muss in `plugin.path` des Workers stehen (`plugin.path=/opt/kafka/plugins`). Starten Sie danach jeden Worker neu.

Im Container

Binden Sie das entpackte Plugin und das Lizenzverzeichnis in den Worker-Container ein, zum Beispiel mit dem offiziellen Image `apache/kafka` (es läuft als `appuser` mit dem Heimatverzeichnis `/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` läuft ebenfalls als `appuser` mit demselben Heimatverzeichnis; sein vorgegebener Plugin-Pfad ist `/usr/share/confluent-hub-components`.

Die Installation prüfen

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

listet `com.toddysoft.connect.integrations.kafka.modbus.ModbusSourceConnector` und `…ModbusSinkConnector` mit ihrer Version.

Aktualisieren

Entfernen Sie das alte Plugin-Verzeichnis, bevor Sie das neue installieren (das Confluent CLI tut das selbst): Liegen zwei Versionen eines Plugins auf `plugin.path`, lädt der Worker eine beliebige. Starten Sie jeden Worker neu.

Lizenz

Der Treiber liest seine Lizenz aus **`.lic`-Dateien in `~/.toddysoft/licenses/` des Benutzers, unter dem der Worker läuft**:

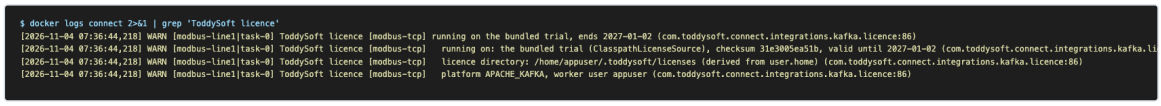
der Worker läuft	die Lizenzdatei gehört nach
als Dienst, Benutzer `kafka`	`~kafka/.toddysoft/licenses/modbus.lic` (zum Beispiel `/home/kafka/.toddysoft/licenses/`)
im Container `apache/kafka` oder `cp-kafka-connect`	ein Host-Verzeichnis, eingebunden unter `/home/appuser/.toddysoft/licenses` (siehe oben)
in einem verteilten Cluster	dasselbe auf **jedem** Worker: Eine Task, die auf einen Worker ohne Lizenz wandert, läuft dort mit der Testlizenz

Der Dateiname ist gleichgültig, nur die Endung `.lic` zählt; eine Datei je Treiber oder eine Lizenz für mehrere Treiber funktioniert beides. Eine neue Lizenz gilt ab dem nächsten Start des Connectors oder der Task (starten Sie den Connector neu). Ohne Lizenz, oder mit einer, die der Treiber ablehnt, gilt die mitgelieferte Testlizenz bis zu ihrem Ablauf; danach lehnt der Treiber seine Verbindungen ab.

Die Lizenz prüfen

Jede Task schreibt beim Start einen **Lizenzbericht** in das Log des Workers, einen Block je Treiber, jede Zeile beginnt mit `ToddySoft licence [modbus-tcp]` (`modbus-tcp` ist die Kennung des Modbus-TCP-Treibers):

```
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
```



Sie finden ihn mit `grep 'ToddySoft licence' <Worker-Log>` (im Container: `docker logs connect 2>&1 | grep 'ToddySoft licence'`). Eine Lizenz wird über ihre **Prüfsumme** benannt, die ersten 12 Zeichen ihres SHA-256: `shasum -a 256 modbus.lic` zeigt, welche Datei gemeint ist. Der Bericht nennt den Treiber mit seinem Protokollcode, `[modbus-tcp]`; die Lizenz nennt ihn `MODBUS`, und eine Lizenz für einen anderen Treiber wird mit `License does not cover the 'MODBUS' driver` abgelehnt.

der Bericht sagt	Bedeutung	was zu tun ist
`licensed` · running on: a licence file	Ihre Lizenz gilt	nichts
`licensed, expires in N days`	sie läuft bald ab	rechtzeitig verlängern
`licensed in its grace period: expired on <date>, refused after <date>`	sie ist abgelaufen; ToddySoft Connect akzeptiert sie noch 30 Tage (Nachfrist)	jetzt verlängern: nach diesem Datum wird die Lizenz abgelehnt
`attention: Support ends in N days (<date>)` / `attention: Support ended on <date>`	Treiberversionen, die nach diesem Datum gebaut wurden, laufen mit dieser Lizenz nicht	den Support verlängern, bevor das Plugin aktualisiert wird
`running on the bundled trial, ends <Datum>` und keine Zeile *found and rejected*	keine Lizenzdatei gefunden	die Datei in das genannte Verzeichnis legen, für den genannten Benutzer
… und `found and rejected: …: PLATFORM_NOT_LICENSED`	die Lizenz deckt Kafka Connect nicht ab	bei ToddySoft eine Lizenz anfordern, die es tut
… `DRIVER_NOT_LICENSED`	es ist die Lizenz eines anderen Treibers	die Modbus-Lizenz verwenden
… `LICENSE_EXPIRED`	sie ist abgelaufen	verlängern
`NOT licensed: …`	keine Lizenz gilt, auch nicht die Testlizenz; der Treiber lehnt jede Verbindung ab	wie oben

Lehnt der Treiber eine Verbindung wegen der Lizenz ab, erscheint derselbe Block einmal, danach wird nur gezählt (`refused N connections since …, licence unchanged`, höchstens alle zehn Minuten), statt das Log zu füllen. Nichts im Log zeigt die Lizenz selbst oder auf wen sie lautet. Die Task schlägt wegen der Lizenz nicht fehl: Der Connector bleibt RUNNING, und seine Datensätze tragen die Ablehnung in `errors`; die Ursache steht im Bericht.

Die Konfigurationsdatei

Beide Connectoren führen eine **Event-Pump-Konfiguration** von ToddySoft Connect aus (Version 2): die Verbindungen, die Variablen unter Aliasnamen und Batches - welche Variablen zusammen gelesen werden, und wann. Das Verzeichnis `etc/` des Archivs enthält ein Beispiel; ein vollständiges:

```
{
  "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 } }
  ]
}
```

- **Verbindungs-URL**: `modbus-tcp://<host>` (Port 502; ein anderer Port als `modbus-tcp://<host>:<port>`). Das Beispiel in `etc/` schreibt den Transport aus, `modbus-tcp:tcp://<host>:502`, was dasselbe ist. Parameter folgen nach dem `?`: `unit-identifier` (die Unit- bzw. Slave-ID, 0-255, Vorgabe 1), `default-payload-byte-order` (siehe unten), `request-timeout-ms` (eine Anfrage als Ganzes, Vorgabe 5000), `max-registers-per-request` (Vorgabe 125), `max-coils-per-request` (Vorgabe 2000), `max-address-gap-to-merge` (Vorgabe 10), `max-concurrent-requests` (Vorgabe 1): `modbus-tcp://192.168.10.20?unit-identifier=3&default-payload-byte-order=LITTLE_ENDIAN_BYTE_SWAP`.
- **Adressen** lauten `<Bereich>:<Register>[:<Typ>]`, die Bereiche sind `holding-register`, `input-register`, `coil` und `discrete-input`: `holding-register:6:INT`, `holding-register:10:DINT` (Register 10-11), `input-register:20:REAL` (Register 20-21), `holding-register:40:STRING(20)` (eine Zeichenkette aus 20 Zeichen; ein `STRING` braucht seine Länge), `coil:3`, `discrete-input:7`. Ohne Typ wird ein Register als `INT` gelesen, eine Coil oder ein Discrete Input als `BOOL`. Registernummern **zählen ab 1**: `holding-register:1` ist das erste Holding-Register, Offset 0 auf der Leitung; zählt die Belegung des Geräts ab 0, addieren Sie eins. Eine Array-Auswahl steht vor dem Typ: `holding-register:100[0..9]:DINT` (zehn DINTs, Register 100-119), `coil:1[0..7]:BOOL` (acht Coils); `holding-register:100[9]:DINT` ist **ein** DINT, das zehnte. Die Schneider-Form `%MW0` / `%M0` wird ebenfalls akzeptiert.
- **Optionen je Variable** in geschweiften Klammern überschreiben die Verbindung für eine Variable: `holding-register:100:INT{unit-id: 5}`, `holding-register:100[0..9]:DINT{unit-id: 5, byte-order: LITTLE_ENDIAN}`. Mit `unit-id` erreichen Sie mehrere Units hinter einer Verbindung, typischerweise eine serielle Leitung hinter einem Modbus-Gateway.
- **Byte-Reihenfolge**: Modbus legt für einen Wert, der breiter als ein Register ist, keine Anordnung fest, daher unterscheiden sich die Geräte. Für einen 32-Bit-Wert `AA BB CC DD`: `BIG_ENDIAN` (Vorgabe, ABCD), `BIG_ENDIAN_BYTE_SWAP` (BADC), `LITTLE_ENDIAN_BYTE_SWAP` (CDAB, "Word Swap", niederwertiges Wort zuerst), `LITTLE_ENDIAN` (DCBA). Eine falsche Einstellung schlägt nicht fehl, sie liefert plausiblen Unsinn.
- **Auslöser**: `timer` (alle `intervalMillis`), `change` (den Batch lesen, wenn sich eine beobachtete Variable ändert, wahlweise unter einer `condition` wie `oldValue == false && value == true`), `subscription` (der Treiber abonniert und reicht jede Änderung weiter; Modbus kennt keine Benachrichtigungen, daher fragt der Treiber dafür zyklisch ab).
- **Zugriff**: `read`-Variablen werden nie geschrieben, `write`-Variablen nie gelesen; ohne `access` beides.
- **Mehrere gleiche Steuerungen**: siehe unten.

Übergeben Sie die Datei dem Connector entweder direkt (`configuration`) oder als Pfad auf **jedem** Worker (`configuration.file`). Eine Konfiguration mit Fehlern weist den Connector ab und nennt jeden Fehler mit seiner Stelle in der Datei und einer Regelkennung.

Zugangsdaten

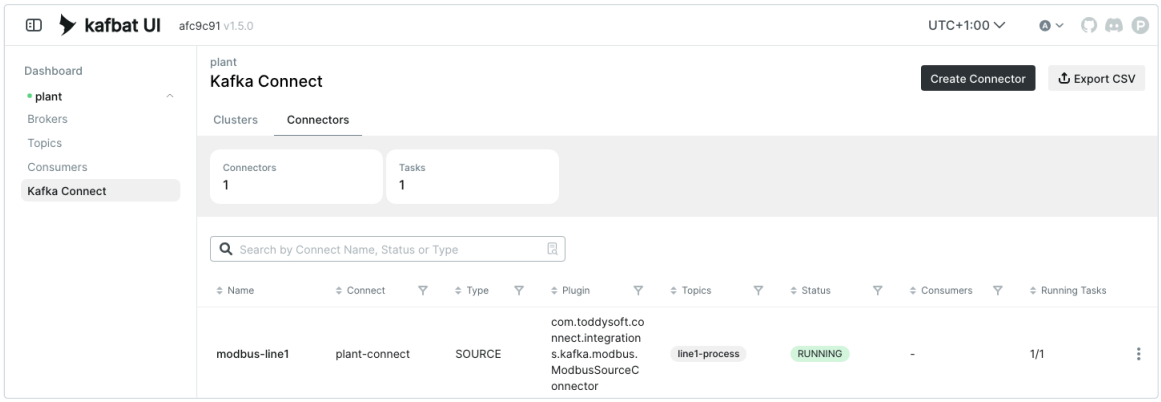
Modbus kennt keine Authentifizierung: Eine Modbus-Verbindung hat kein `credential`, und der Connector braucht keine Eigenschaften `secrets.<name>`. Jeder Rechner, der Port 502 eines Geräts erreicht, kann es lesen und beschreiben; betreiben Sie die Geräte daher in einem Netz, das nur die Worker und andere berechtigte Clients erreichen.

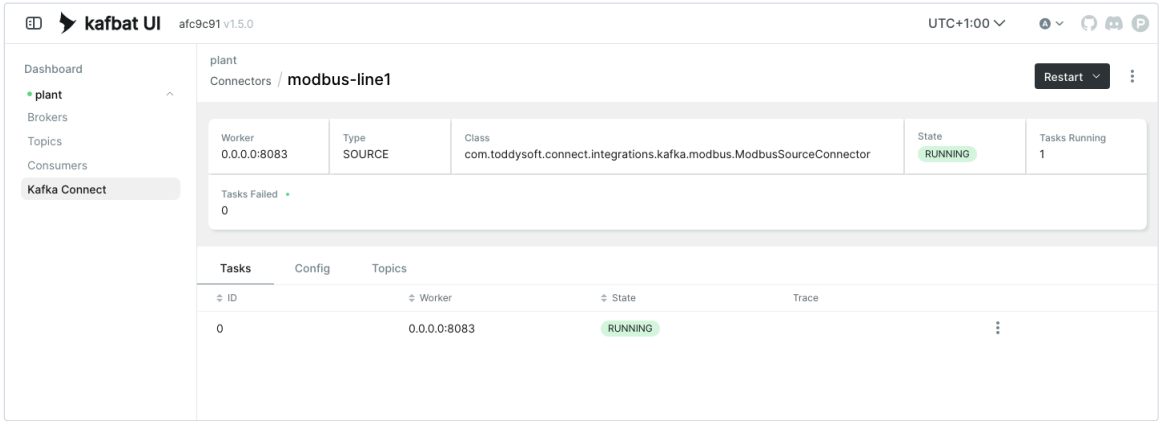
Der Source-Connector

```
{
  "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` legt ihn an.

Eigenschaft	Vorgabe	Bedeutung
`configuration` / `configuration.file`	-	die Konfiguration, direkt oder als Pfad auf jedem Worker; genau eines davon
`batches`	alle	die auszuführenden Batches, durch Kommas getrennt
`batches.topics`	`{}`	JSON-Objekt: Batch zu Topic
`topic.default`	-	Topic jedes Batches, der nicht in `batches.topics` steht
`tasks.max`	1	Batches (und Verbindungen, siehe unten) werden auf höchstens so viele Tasks verteilt





Die Bildschirmfotos zeigen kafka-ui (kafbat UI), eine freie Weboberfläche für Kafka und Kafka Connect; jedes Werkzeug, das Connectoren und Topics zeigt, eignet sich, unter Confluent Platform zum Beispiel Confluent Control Center. Sie entstanden am Modbus-TCP-Server einer S7-1200, von dem drei Holding-Register-Adressen (BOOL, INT, DINT) einmal pro Sekunde gelesen werden.

Datensätze

Ein Datensatz je Lesevorgang. Sein Wert ist ein `Struct`:

Feld	Typ	
`batch`	STRING	der Batch
`connection`	STRING	die Verbindung
`timestamp`	INT64	der Zeitpunkt des Lesens, Millisekunden seit der Epoche
ein Feld je Variablen-Alias	siehe Typtabelle	`null`, wenn die Variable nicht gelesen werden konnte
`errors`	MAP<STRING,STRING>	Alias zu Grund, für jede Variable, die nicht gelesen werden konnte

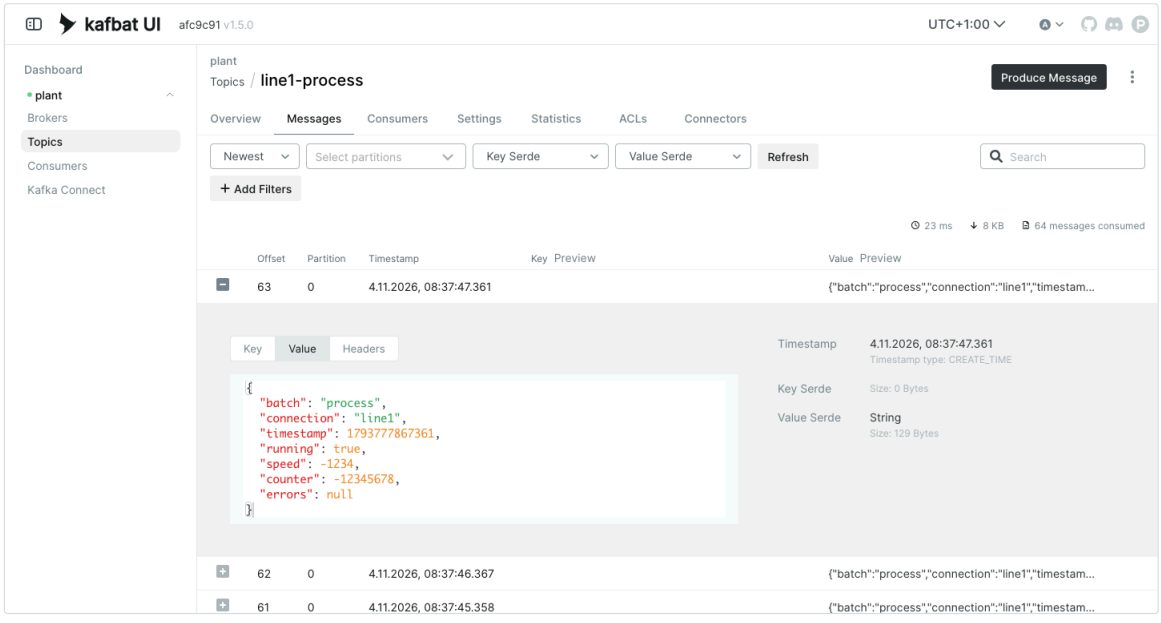
Der Datensatz hat **keinen Schlüssel**; seine Source-Partition ist `{batch, connection}`, sein Offset `{timestamp}`. Mit der Konfiguration oben lautet ein Datensatz des Batches `process`, als JSON ohne Schema:


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


Typ in der Adresse	Connect-Typ	
BOOL	BOOLEAN	
SINT	INT8	
BYTE, USINT, INT	INT16	vorzeichenlose 8-Bit-Typen erweitert
WORD, UINT, DINT	INT32	vorzeichenlose 16-Bit-Typen erweitert
DWORD, UDINT, LINT	INT64	vorzeichenlose 32-Bit-Typen erweitert
LWORD, ULINT	Decimal (Skala 0)	vorzeichenlos 64 Bit, exakt
REAL	FLOAT32	
LREAL	FLOAT64	
CHAR, WCHAR, STRING, WSTRING	STRING	
TIME, LTIME	INT64	Millisekunden
TIME_OF_DAY, LTIME_OF_DAY	INT64	Millisekunden seit Mitternacht
DATE, LDATE	INT32	Tage seit der Epoche
DATE_AND_TIME, LDATE_AND_TIME	Timestamp	
ein Array	ARRAY of STRING	jedes Element als Text

Modbus legt keine Kodierung für Datum und Uhrzeit fest; der Treiber liest sie als vorzeichenlose Zähler - `TIME` in Millisekunden, `DATE` und `DATE_AND_TIME` in Sekunden seit der Epoche, `TIME_OF_DAY` in Millisekunden seit Mitternacht, die `L`-Varianten ebenso in Nanosekunden. Kodiert ein Gerät sie anders, lesen Sie die Register als Ganzzahltyp und rechnen im weiteren Verlauf um.

Ein Feld erhält seinen Typ beim **ersten Lesen** des Batches, und jedes Feld ist optional: Eine Variable, die beim ersten Lesen nicht gelesen werden konnte, ist danach ein STRING - starten Sie den Connector neu, sobald das Gerät antwortet.



Konverter und Datenformate

Der Connector erzeugt Connect-Datensätze vom Typ `Struct` mit Schema; der Konverter des Workers bestimmt die Bytes im Topic:

- `org.apache.kafka.connect.json.JsonConverter` mit `value.converter.schemas.enable=true` (Schema und Nutzdaten) oder `false` (nur Nutzdaten);
- `io.confluent.connect.avro.AvroConverter` oder `io.confluent.connect.protobuf.ProtobufConverter` mit Confluents Schema Registry (`value.converter.schema.registry.url=http://registry:8081`), auf einem Worker, der diese Konverter hat (Confluent Platform hat sie).

Setzen Sie den Konverter je Connector (`"value.converter": "…"`) oder für den Worker. Jeder Lesevorgang eines Batches hat dasselbe Schema; eine zum Batch hinzugefügte Variable ändert es.

Der Sink-Connector

Schreibt die Felder jedes Datensatzes in die Variablen einer Verbindung, über den 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"
  }
}
```

Eigenschaft	Vorgabe	Bedeutung
`connection`	-	die Verbindung, in die geschrieben wird
`topics`	-	die gelesenen Topics
`max.retries`	3	wie oft ein Schreibvorgang, den das Gerät abgelehnt hat, wiederholt wird
`retry.backoff.ms`	1000	die erste Wartezeit vor einer Wiederholung, jedes Mal verdoppelt
`retry.backoff.max.ms`	30000	die längste Wartezeit

Ein Datensatz ist ein `Struct` (mit einem Konverter, der ein Schema mitführt, zum Beispiel JSON mit Schema oder Avro); jedes Feld nennt einen Variablen-Alias der Verbindung, sein Typ folgt der Typtabelle. Modbus schreibt nur **Coils und Holding-Register**; Input-Register und Discrete Inputs sind laut Protokoll nur lesbar, geben Sie ihren Variablen daher `"access": "read"`. Ein Feld, das die Verbindung nicht kennt, oder eine Variable mit dem Zugriff `read` wird abgewiesen: Der Datensatz geht ungeschrieben an die Fehlerbehandlung von Kafka Connect (eine Dead Letter Queue mit `errors.tolerance=all` und `errors.deadletterqueue.topic.name`). Ein Schreibvorgang, den das Gerät ablehnt, wird wiederholt und dann ebenso gemeldet. Ein wegen der Lizenz abgelehnter Schreibvorgang wird nicht wiederholt.

Mehrere gleiche Steuerungen

Ein Batch mit `connectionIds` läuft einmal je Verbindung - meist liest er einen Variablensatz, den jede Steuerung hat:

```
{
  "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 } }
  ]
}
```

Jede Presse wird als eigene Einheit gelesen, verteilt auf `min(Einheiten, tasks.max)` Tasks: Mit `tasks.max` 3 in einem Cluster aus drei Workern liest jeder Worker eine Presse. Die Datensätze behalten den Batch `line-speed` und nennen die Verbindung, sodass ein Topic alle Pressen enthält, partitioniert nach Presse.

Im **verteilten Modus** verschiebt Kafka Connect Tasks bei einem Rebalance zwischen Workern; eine verschobene Task verbindet sich von ihrem neuen Worker aus und schreibt dort ihren Lizenzbericht. Die Tasks eines gestoppten Workers wandern erst nach bis zu fünf Minuten (`scheduled.rebalance.max.delay.ms`): so lange wartet Kafka Connect, ob der Worker zurückkommt. Im Standalone-Modus läuft alles auf einem Worker.

Mehrere Treiber-Plugins auf einem Worker

Jeder Treiber ist ein eigenes Plugin, und sie liegen nebeneinander auf einem Worker (`/opt/kafka/plugins/toddysoft-kafka-connect-modbus`, `…-s7`, …). **Verwenden Sie die Connector-Klassen des jeweiligen Treibers** - hier `…kafka.modbus.ModbusSourceConnector`: Ein Worker lädt eine Connector-Klasse nur aus einem Plugin. Die allgemeinen Klassen `PlcSourceConnector` / `PlcSinkConnector` gibt es in jedem Plugin; sie sind nicht zur Verwendung gedacht. Eine Task, die den Treiber ihrer URL nicht findet, startet nicht und nennt die richtige Klasse. Innerhalb eines Plugins teilen sich ein Source- und ein Sink-Connector zum selben Gerät eine Verbindung.

Fehlersuche

Symptom	Ursache	was zu tun ist
das Plugin fehlt in `GET /connector-plugins`	nicht auf `plugin.path`, oder der Worker wurde nicht neu gestartet	`plugin.path` und Verzeichnis prüfen, jeden Worker neu starten
das Plugin fehlt in der Liste, und das Worker-Log zeigt `Failed to discover SourceConnector in …` mit der Ursache `UnsupportedClassVersionError … (class file version 61.0)`	der Worker läuft mit Java 11 oder älter	den Worker mit Java 17 oder neuer betreiben
eine Task schlägt mit `driver-missing` fehl	die allgemeine oder die Klasse eines anderen Treibers	`…kafka.modbus.ModbusSourceConnector` / `ModbusSinkConnector` verwenden
der Connector wird mit Regelkennung und Stelle abgewiesen	die Konfiguration enthält einen Fehler	die Datei an jeder genannten Stelle korrigieren
RUNNING, aber jeder Datensatz hat die Variablen `null` und den Grund in `errors`	das Gerät lehnt ab oder ist nicht erreichbar	`errors` lesen: ein Modbus-Antwortcode, eine Lizenz
`errors` nennt `License validation failed`	keine Lizenz gilt	siehe Die Lizenz prüfen
`Connection refused` auf Port 502 im Worker-Log	der Modbus-TCP-Server des Geräts ist aus, ein anderer Port, oder eine Firewall	Modbus TCP im Gerät einschalten, Port und Firewalls prüfen
der Grund einer Variablen in `errors` ist `INVALID_ADDRESS`	das Gerät hat mit *Illegal Data Address* geantwortet (Exception 2): Das Register liegt außerhalb seiner Belegung, oder das Array reicht über ihr Ende hinaus. Zu einer Anfrage zusammengefasste Variablen teilen sich die Antwort, eine korrekte Variable kann sie also für eine benachbarte melden	jede Adresse und Array-Länge des Batches mit der Belegung des Geräts vergleichen; `max-address-gap-to-merge` verkleinern
die Werte stammen aus dem benachbarten Register	Registernummern zählen ab 1, die Belegung des Geräts ab 0	zu den Offsets der Belegung eins addieren
`UNSUPPORTED`	das Gerät hat mit *Illegal Function* geantwortet (Exception 1): Es bedient diesen Bereich oder diesen Schreibvorgang nicht	einen Bereich verwenden, den das Gerät laut Belegung bedient
`NOT_FOUND`, oder keine Daten von einem Gerät hinter einem Modbus-Gateway	das Gateway hat unter dieser Unit kein Gerät erreicht (Exception 11): Der Unit Identifier ist falsch	`unit-identifier` in der URL oder `{unit-id: N}` an der Variablen setzen
`REMOTE_ERROR`, und das Worker-Log warnt, dass eine `response from unit N … was rejected`	eine Antwort kam von einer anderen Unit als der angefragten	die Unit-Zuordnung des Gateways prüfen; für ein Gerät, das die Unit nicht zurückgibt, `verify-unit-identifier=false`
plausible, aber falsche Werte: absurde Gleitkommazahlen, Ganzzahlen mit vertauschten Hälften	das Gerät verwendet eine andere Byte-Reihenfolge	`default-payload-byte-order` in der URL oder `{byte-order: …}` an der Variablen setzen (siehe Die Konfigurationsdatei)

Symptom	Ursache	was zu tun ist
`REMOTE_BUSY`, und das Worker-Log warnt, dass innerhalb von `request-timeout-ms` keine Antwort kam	die Anfrage wurde nicht rechtzeitig fertig	`request-timeout-ms` erhöhen oder weniger Variablen in den Batch nehmen
merkwürdiges Verhalten nach einer Aktualisierung	zwei Kopien des Plugins auf `plugin.path`: Der Worker listet es einmal und protokolliert nichts, man sieht also nicht, welche Kopie läuft	das alte Verzeichnis entfernen, neu starten

Mehr Einzelheiten liefert eine höhere Protokollstufe des Treibers: `curl -X PUT -H 'Content-Type: application/json' --data '{"level":"DEBUG"}' http://localhost:8083/admin/loggers/com.toddysoft` (und danach zurück auf `INFO`). Jede Ablehnung durch ein Modbus-Gerät protokolliert der Treiber außerdem mit WARN, mit dem Namen der Exception und ihrer Bedeutung.

Versionshistorie

1.3.0 — 2026-11-13

Basiert auf ToddySoft Connect 1.3.0.

- **Behoben:** Schreibzugriffe auf erweiterte Register (`6x`) funktionieren; bisher scheiterte jeder, bevor er gesendet wurde. Große zusammengefasste Schreibzugriffe halten die Grenze von Modbus pro Anfrage ein.
- **Neu:** der Modbus-Connector als Produkt für selbst betriebene Kafka-Connect-Worker - dieses Handbuch, ein Komponentenarchiv für Confluent Hub neben der Plugin-ZIP, beide von jedem Release geprüft; ein Lizenzbericht je Treiber im Log des Workers; unterstützte Versionen Apache Kafka 3.9.x und 4.0-4.4, Confluent Platform 7.9 und 8.0-8.4.