ToddySoft Connect EtherNet/IP - Apache Kafka Connect

Ein Kafka-Connect-**Source- und Sink-Connector** für **Logix**-Steuerungen von Allen-Bradley - ControlLogix, CompactLogix und verwandte - auf Basis des EtherNet/IP-Treibers von ToddySoft Connect. Er liest Variablen über EtherNet/IP und CIP unter den **Symbolnamen aus dem Logix-Programm**, steuerungs- oder programmweit, und veröffentlicht jeden Lesevorgang als einen Kafka-Datensatz; der Sink schreibt Felder von Datensätzen in Variablen. EtherNet/IP-Adapter ohne benannte Variablen (E/A-Koppler, Antriebe) erreicht er über ihre CIP-Objekte.

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 keine Steuerung 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 oder bei einer Wertänderung (mit Bedingung)	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 Steuerungen** (TCP-Port 44818).
- **In der Steuerung**: EtherNet/IP aktiviert (auf CompactLogix- und ControlLogix-CPUs mit integriertem Ethernet standardmäßig an; eine ControlLogix mit separater Karte 1756-ENBT / 1756-EN2T braucht diese Karte konfiguriert) und *Allow external access* in den Programmeigenschaften aktiviert (Studio 5000 / RSLogix 5000): Ist es deaktiviert, kommt die Verbindung zustande, die Variablen sind aber nicht lesbar. Zugangsdaten gibt es keine (siehe Zugangsdaten).
- Eine **ToddySoft-Lizenz für EtherNet/IP** 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-eip-<version>.zip` - im Komponentenarchiv-Format von Confluent Hub, für Confluents Installationsprogramm;
- `toddysoft-kafka-connect-eip-<version>-plugin.zip` - ein einfaches Plugin-Verzeichnis.

Mit dem Confluent CLI (Confluent Platform)

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

listet `com.toddysoft.connect.integrations.kafka.eip.EipSourceConnector` und `…EipSinkConnector` 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**:

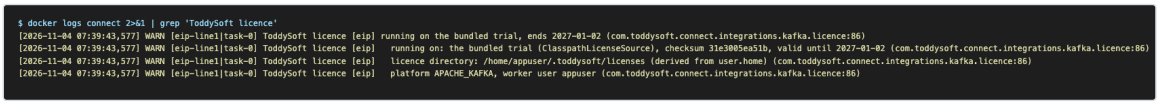
der Worker läuft	die Lizenzdatei gehört nach
als Dienst, Benutzer `kafka`	`~kafka/.toddysoft/licenses/eip.lic` (zum Beispiel `/home/kafka/.toddysoft/licenses/`)
im Container `apache/kafka` oder `cp-kafka-connect`	ein Host-Verzeichnis, eingebunden unter `/home/appuser/.toddysoft/licenses` (siehe oben)
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 [eip]`:

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

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 EtherNet/IP-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": "eip://192.168.10.20",
      "tags": {
        "running":  "Program:Process.Running",
        "speed":    "Program:Process.Speed",
        "counter":  "Counter",
        "setpoint": { "address": "Setpoint:REAL", "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**: `eip://<host>` (Port 44818). Backplane und Slot geben an, wo die Steuerung in ihrem Chassis sitzt, vom Ethernet-Modul aus gesehen, das die Verbindung erreicht: `backplane` und `slot` (Vorgabe 1 und 0, passend für eine CompactLogix und eine ControlLogix mit der CPU in Steckplatz 0): `eip://192.168.10.20?slot=2`. Das Routing über mehrere Backplanes oder durch eine Bridge-Karte nimmt `communication-path` (Paare aus 1 = Backplane und Steckplatz oder 2 = Ethernet und IP-Adresse): `eip://192.168.10.20?communication-path=1,4,2,192.168.20.5,1,0`.
- **Adressen**, symbolisch, wie das Logix-Programm sie benennt: eine steuerungsweite Variable `MyTag`, eine programmweite Variable `Program:MainProgram.LocalTag`, ein Strukturelement `MyStruct.Field`, ein Array-Element `MyArray[5]`, eine Zelle eines mehrdimensionalen Arrays `MyArray[2,3]`, Auswahlen auf mehreren Segmenten `Matrix[0].Row[1]`. Ein Array ohne Index wird ganz gelesen (ein gepacktes `BOOL[32]` als 32 BOOLs); Zeichenketten werden ganz gelesen, eigene Zeichenkettentypen des Projekts (`STRING_16`, …) eingeschlossen. Eine programmweite Variable braucht das Präfix `Program:<name>.`, eine steuerungsweite darf es nicht haben. Groß- und Kleinschreibung spielt keine Rolle. **Eine Struktur (UDT) wird nicht als Ganzes gelesen, ebenso wenig ein Array von UDTs**: Adressieren Sie ihre Elemente.
- **CIP-Objekte**, auf einem EtherNet/IP-Adapter ohne benannte Variablen: `@assembly/104` (die ganze Assembly, als Rohbytes), `@assembly/104+2:UINT` (der UINT ab Byte 2), `@assembly/104+3.5:BOOL` (Bit 5 von Byte 3), `@1/1/7:STRING` (der Produktname aus dem Identity-Objekt). Ein Schreibvorgang auf eine Objektadresse ersetzt das ganze Attribut.
- **Auslöser**: `timer` (alle `intervalMillis`), `change` (den Batch lesen, wenn sich eine beobachtete Variable ändert, wahlweise unter einer `condition` wie `oldValue == false && value == true`). Der EtherNet/IP-Treiber abonniert nicht; `subscription` ist für diesen Connector daher nicht geeignet.
- **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

Es gibt keine einzurichten: EtherNet/IP, wie dieser Treiber es spricht, kennt keine Authentifizierung, daher hat eine Verbindung kein `credential`, und die Connectoren brauchen kein `secrets.<name>`. Der Zugriff wird in der Steuerung geregelt: *Allow external access* in den Programmeigenschaften, *External Access* jeder Variablen in Studio 5000 sowie Betriebsart und Schlüsselschalter der Steuerung.

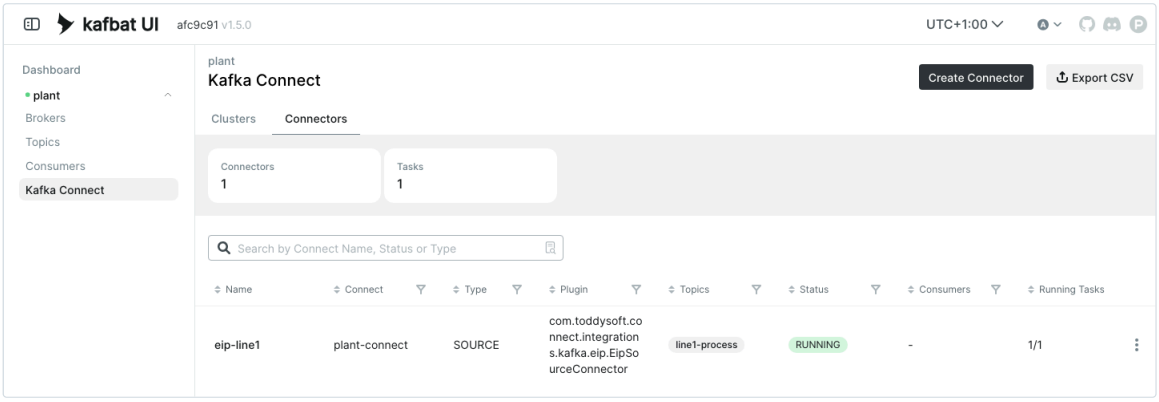
Kafka protokolliert die Konfigurationswerte dieser Connectoren trotzdem nie. Beschränken Sie das Config-Topic des Workers mit ACLs.

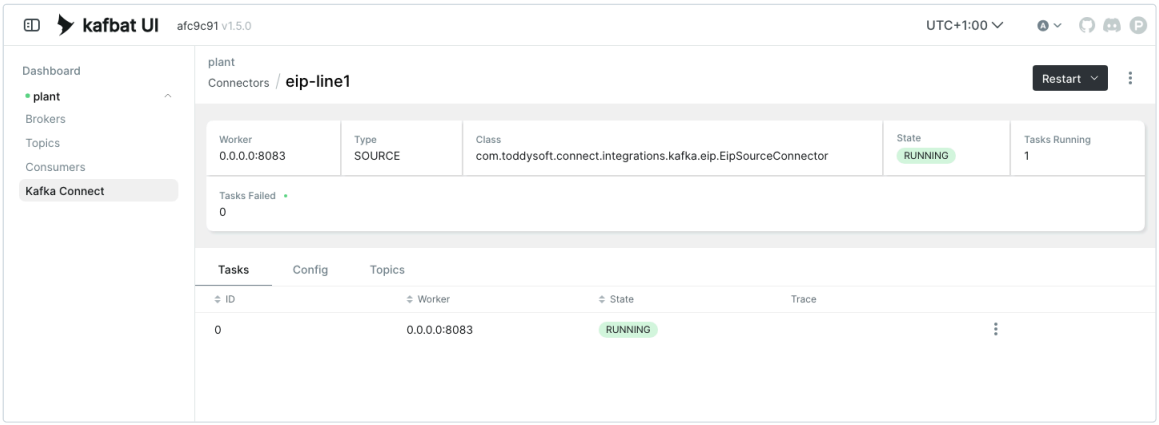
Der Source-Connector

```
{
  "name": "eip-line1",
  "config": {
    "connector.class": "com.toddysoft.connect.integrations.kafka.eip.EipSourceConnector",
    "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 @eip-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
`secrets.<name>`	-	der Wert von `{{name}}` (für EtherNet/IP nicht nötig)
`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 an einer CompactLogix 5370, die die Programmvariablen `Program:TestProgram.g_b1`, `g_s16` und `g_s32` einmal pro Sekunde liest.

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

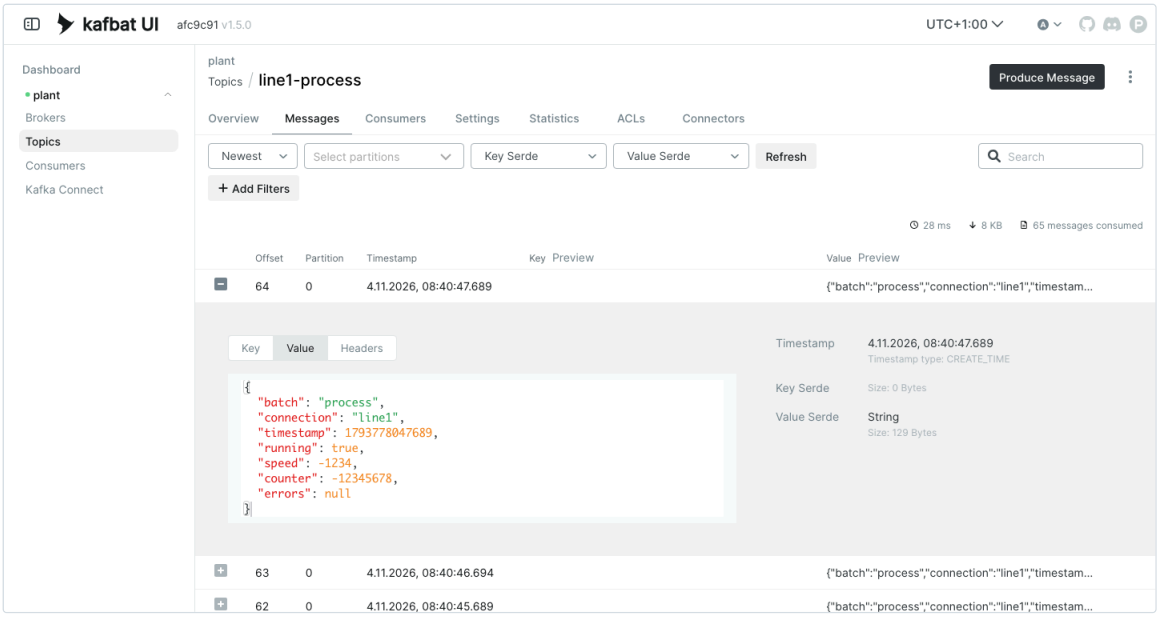
Der Datensatz hat **keinen Schlüssel**; seine Source-Partition ist `{batch, connection}`, sein Offset `{timestamp}`. Von der CompactLogix oben gelesen, mit den Aliasnamen `running`, `speed` und `counter` für einen BOOL, einen INT und einen DINT:


```
{"batch":"process","connection":"line1","timestamp":…,"running":true,"speed":-1234,"counter":-12345678,"errors":null}
```


Typ in der Steuerung	Connect-Typ	
BOOL	BOOLEAN	
SINT	INT8	
USINT, BYTE, INT	INT16	vorzeichenlose 8-Bit-Typen erweitert
UINT, WORD, DINT	INT32	vorzeichenlose 16-Bit-Typen erweitert
UDINT, DWORD, LINT	INT64	vorzeichenlose 32-Bit-Typen erweitert
ULINT, LWORD	Decimal (Skala 0)	vorzeichenlos 64 Bit, exakt
REAL	FLOAT32	
LREAL	FLOAT64	
STRING, ein Zeichenkettentyp des Projekts	STRING	
ein Array (auch ein gepacktes BOOL-Array)	ARRAY of STRING	jedes Element als Text
eine CIP-Objektadresse ohne Typ	BYTES	die Rohdaten
eine Struktur (UDT), ein Array von UDTs	-	wird nicht gelesen (`UNSUPPORTED`); ihre Elemente adressieren

Eine Logix-Steuerung hat keinen eigenen Datums- oder Zeittyp: Solche Werte sind im Programm DINT- oder LINT-Zähler und kommen als INT32 oder INT64 an.

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 die Steuerung 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": "eip-line1-setpoints",
  "config": {
    "connector.class": "com.toddysoft.connect.integrations.kafka.eip.EipSinkConnector",
    "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 die Steuerung 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. Der Treiber schreibt ein Feld als das, was es ist - BOOLEAN als BOOL, INT16 als INT, INT32 als DINT, INT64 als LINT, FLOAT32 als REAL, FLOAT64 als LREAL, STRING als STRING -, und die Steuerung wandelt nichts um: Bei einer Variablen anderen Typs beenden Sie ihre Adresse mit dem Typ (`Setpoint:REAL`, `Limit:UINT`), dann wird der Wert in diesen umgewandelt. Strukturen und einzelne Elemente gepackter BOOL-Arrays sind nicht schreibbar; schreiben Sie die Elemente einer Struktur. 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 die Steuerung 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": "Program:Press.Speed", "count": "Program:Press.Count" } },
  "connections": [
    { "id": "press-1", "url": "eip://192.168.10.31", "tagSets": ["press"] },
    { "id": "press-2", "url": "eip://192.168.10.32", "tagSets": ["press"] },
    { "id": "press-3", "url": "eip://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-eip` , `…-s7` , …). **Verwenden Sie die Connector-Klassen des jeweiligen Treibers** - hier `…kafka.eip.EipSourceConnector` : 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 zur selben Steuerung 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.eip.EipSourceConnector` / `EipSinkConnector` 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`	die Steuerung lehnt ab oder ist nicht erreichbar	`errors` und das Worker-Log lesen: eine Adresse, eine abgelehnte Verbindung, eine Lizenz
`errors` nennt `License validation failed`	keine Lizenz gilt	siehe Die Lizenz prüfen
ForwardOpen abgelehnt mit dem CIP-Status `0x01`	falsche `backplane` / `slot` für das Chassis, oder die Steuerung hat keine freien Verbindungen (eine offene Studio-5000-Sitzung zählt mit)	den Aufbau des Chassis prüfen, andere Clients schließen
ForwardOpen abgelehnt mit dem CIP-Status `0x09`	`communication-path` entspricht keiner tatsächlichen Route im Chassis	weglassen und den Pfad aus `backplane` und `slot` bilden lassen, oder korrigieren
`errors` einer Variablen nennt `NOT_FOUND`	keine solche Variable, kein solches Element - oft der Geltungsbereich: eine programmweite Variable ohne `Program:<name>.` oder eine steuerungsweite mit	Name, Geltungsbereich und Index in Studio 5000 prüfen
`INVALID_ADDRESS`	ein Bereich reicht über das Ende des Arrays hinaus	ihn auf die Grenzen des Arrays kürzen
`UNSUPPORTED` beim Lesen	eine Struktur (UDT) oder ein Array von UDTs; oder ein Gerät ohne benannte Variablen (ein EtherNet/IP-Adapter)	die Elemente adressieren; auf einem Adapter CIP-Objektadressen verwenden (`@assembly/<n>`)
`ACCESS_DENIED` beim Schreiben	*External Access* der Variablen ist *Read Only* (oder sie ist eine Konstante), oder Betriebsart bzw. Schlüsselschalter der Steuerung verbieten es	*External Access* der Variablen in Studio 5000 oder die Betriebsart ändern
`INVALID_DATATYPE` beim Schreiben	der Typ des Werts weicht vom Typ der Variablen ab	die Adresse mit dem Typ der Variablen beenden (`Setpoint:REAL`)

Symptom	Ursache	was zu tun ist
ein Schreibvorgang auf eine Struktur oder ein einzelnes Element eines gepackten BOOL-Arrays wird abgelehnt	Aggregate sind nicht schreibbar; ein einzelnes BOOL eines gepackten Worts bräuchte ein Lesen-Ändern-Schreiben	stattdessen die Elemente der Struktur schreiben
die Verbindung kommt zustande, aber keine Variable ist lesbar	*Allow external access* ist in den Programmeigenschaften deaktiviert	in Studio 5000 aktivieren
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 - den CIP-Status hinter einer Ablehnung protokolliert er auf DEBUG: `curl -X PUT -H 'Content-Type: application/json' --data '{"level":"DEBUG"}' http://localhost:8083/admin/loggers/com.toddysoft` (und danach zurück auf `INFO`).

Versionshistorie

1.3.0 — 2026-11-13

Basiert auf ToddySoft Connect 1.3.0.

- **Geändert:** `backplane` und `communication-path` wirken jetzt beim Verbindungsaufbau; ToddySoft Connect hat beide bisher ignoriert. Führt der Weg zu keiner Steuerung, scheitert der Verbindungsaufbau jetzt mit einer Fehlermeldung, statt dass der Connector mit dem Ethernet-Modul selbst spricht. Prüfen Sie beide nach dem Update.
- **Neu:** der EtherNet/IP-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.