ToddySoft Connect ctrlX - Apache Kafka Connect

Ein Kafka-Connect-**Source- und Sink-Connector** für Steuerungen Bosch Rexroth ctrlX CORE auf Basis des ctrlX-Treibers von ToddySoft Connect. Er liest Knoten des ctrlX Data Layer - SPS-Variablen, Achszustände, Systemmetriken - über die REST-API des Data Layer per HTTPS, mit einem ctrlX-Benutzerkonto, und veröffentlicht jeden Lesevorgang als einen Kafka-Datensatz; der Sink schreibt Felder von Datensätzen in Knoten.

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, 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

	dieser Connector	Connector von Apache PLC4X 1.0.0
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 443, HTTPS).
- **In der Steuerung**: der laufende Data-Layer-Dienst und ein **Benutzerkonto** für den Connector (siehe Zugangsdaten). Zum Lesen braucht es die Berechtigung *Data Layer Global → View resources* (`rexroth-automation.datalayer.r`); *Data Layer → View* ist es nicht. ctrlX vergibt Lesen und Schreiben als getrennte Berechtigungen: Für den Source-Connector genügt ein Konto, das nur lesen darf, der Sink braucht ein Konto, das schreiben darf.
- Eine **ToddySoft-Lizenz für ctrlX** 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-ctrlx-<version>.zip` - im Komponentenarchiv-Format von Confluent Hub, für Confluents Installationsprogramm;
- `toddysoft-kafka-connect-ctrlx-<version>-plugin.zip` - ein einfaches Plugin-Verzeichnis.

Mit dem Confluent 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
```

Das CLI installiert aus der lokalen Datei, ohne Internetzugang, nach `<plugin-directory>/toddysoft-kafka-connect-ctrlx` 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-ctrlx-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 ctrlx
```

listet `com.toddysoft.connect.integrations.kafka.ctrlx.CtrlxSourceConnector` und `…CtrlxSinkConnector` 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**:

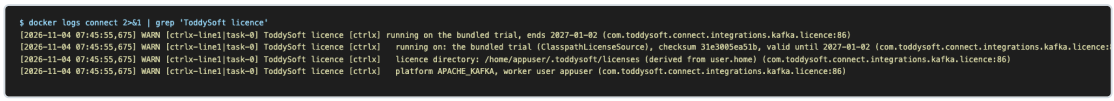
der Worker läuft	die Lizenzdatei gehört nach
als Dienst, Benutzer `kafka`	`~kafka/.toddysoft/licenses/ctrlx.lic` (zum Beispiel `/home/kafka/.toddysoft/licenses/`)
im Container `apache/kafka` oder `cp-kafka-connect`	ein Host-Verzeichnis, eingebunden unter `/home/appuser/.toddysoft/licenses` (siehe oben)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



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 ctrlx.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

der Bericht sagt	Bedeutung	was zu tun ist
`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 ctrlX-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 mit ihren Zugangsdaten, 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": "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 } }
  ]
}
```

- **Verbindungs-URL**: `ctrlx://<host>:443` (443 ist die Vorgabe und kann entfallen). Weitere Parameter werden als Abfrage angehängt: `request-timeout-ms` (eine HTTPS-Anfrage, Vorgabe 5000), `base-path` (die REST-API des Data Layer, Vorgabe `/automation/api/v2`) und die Zertifikatsparameter unten: `ctrlx://192.168.10.20:443?request-timeout-ms=10000`. Das Konto steht nie in der URL.
- **Adressen** sind Knotenpfade des Data Layer, durch Schrägstriche getrennt, wie ctrlX sie zeigt: Eine SPS-Variable ist `plc/app/<Applikation>/sym/<Programm>/<Variable>`, zum Beispiel `plc/app/Application/sym/PLC_PRG/counter`; ein Element einer Struktur `plc/app/Application/sym/PLC_PRG/g_simple/dat`; die Istposition einer Achse `motion/axs/<Achse>/state/values/actual/pos`; eine Systemmetrik `framework/metrics/system/cpu-utilisation-percent`. Ein führender `/` ist wahlweise. Ein Muster wie `framework/metrics/system/**` dient dem Durchsuchen und ist keine Variablenadresse.
- **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; die Steuerung veröffentlicht Änderungen höchstens alle `publish-interval-ms`, ein URL-Parameter, Vorgabe 1000).
- **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.

Das Zertifikat der Steuerung

Eine ctrlX CORE antwortet per HTTPS mit einem **selbstsignierten Zertifikat**, das auf keine öffentliche CA zurückgeht und das Gerät statt seiner Adresse nennt. Der Treiber prüft es (`tls.verify`, standardmäßig an - lassen Sie es an; es prüft auch den Hostnamen) und lehnt ein Zertifikat ab, das er nicht prüfen kann: Das Log des Workers nennt seinen SHA-256-Fingerabdruck (`SHA-256 fingerprint: a3:f8:…:26:34`), und sobald die Verbindung gescheitert ist, tragen die Datensätze den Grund in `errors`. Vergleichen Sie diesen Fingerabdruck mit der Steuerung (ihre Weboberfläche zeigt das Zertifikat) und hinterlegen Sie ihn in der URL:

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

Ein hinterlegter Fingerabdruck nimmt genau die genannten Zertifikate an und hat Vorrang vor `tls.verify`; der Hostname wird dann nicht geprüft, denn der Fingerabdruck bezeichnet die Steuerung bereits. **Eine ctrlX CORE kann mehr als ein Zertifikat vorzeigen**, je Verbindung zufällig eines: Hinterlegen Sie jedes, durch Kommas getrennt. Eine Verbindung, die eines erwischt, das noch nicht hinterlegt ist, wird mit `TLS pin check failed: peer certificate SHA-256 fingerprint <Wert> is not in the pinned list` abgewiesen - ergänzen Sie diesen Wert in der Liste, statt sie zu ersetzen.

Ein Zertifikat, das eine CA signiert hat, der das Java des Workers vertraut, braucht keinen hinterlegten Fingerabdruck, sofern es die Adresse in der URL nennt; einen eigenen Truststore-Parameter hat der ctrlX-Treiber nicht.

Zugangsdaten

Zugangsdaten stehen nie in der URL oder in der Datei selbst. Jedes geheime Feld ist ein **Verweis** `{{name}}`, und die Connector-Eigenschaft `secrets.<name>` liefert den Wert - am besten über einen **Config Provider** von Kafka, damit der Wert auch nicht in der Connector-Konfiguration steht. Aktivieren Sie den Datei-Provider auf jedem Worker:

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

und verweisen Sie auf eine Properties-Datei, die nur der Benutzer des Workers lesen darf:

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

die Steuerung	`method`	anzugeben
ctrlX CORE (jede Anfrage an den Data Layer braucht eine Anmeldung)	`username-password`	`username`, `password`

Der Benutzername ist **Pflicht**: Der Data Layer kennt keinen anonymen Zugang, und eine Verbindung ohne Zugangsdaten wird abgewiesen. Das Passwort muss ein Verweis sein; den Benutzernamen dürfen Sie auch unverändert in die Datei schreiben. Verwenden Sie für den Connector ein eigenes Konto mit nicht mehr Rechten, als er braucht (siehe Voraussetzungen).

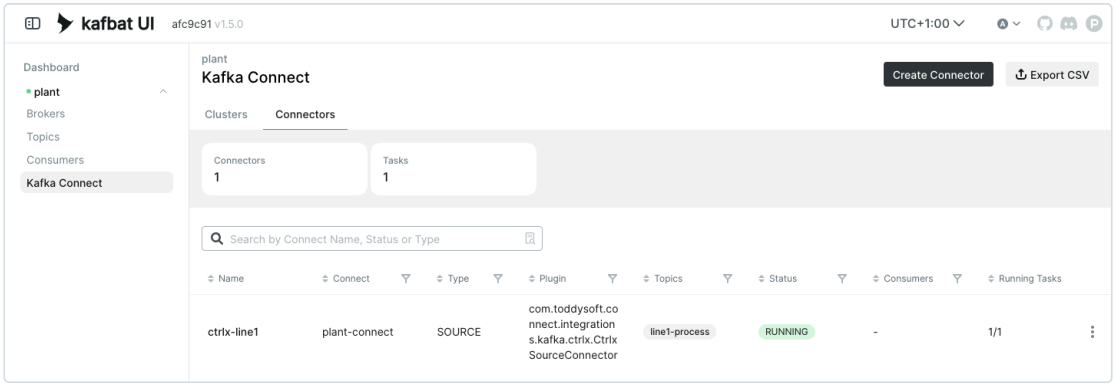
Kafka protokolliert die Konfigurationswerte dieser Connectoren nie, und der Worker schreibt den Platzhalter `${file:…}`, nicht den Wert, in sein Config-Topic. Beschränken Sie das Config-Topic trotzdem mit ACLs.

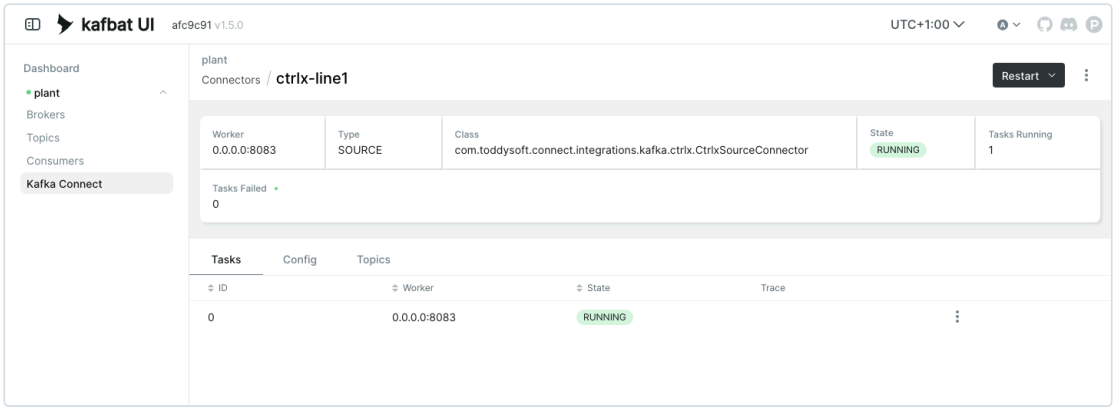
Der Source-Connector

```
{
  "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` 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}}`
`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 ctrlX CORE, die `plc/app/Application/sym/PLC_PRG/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

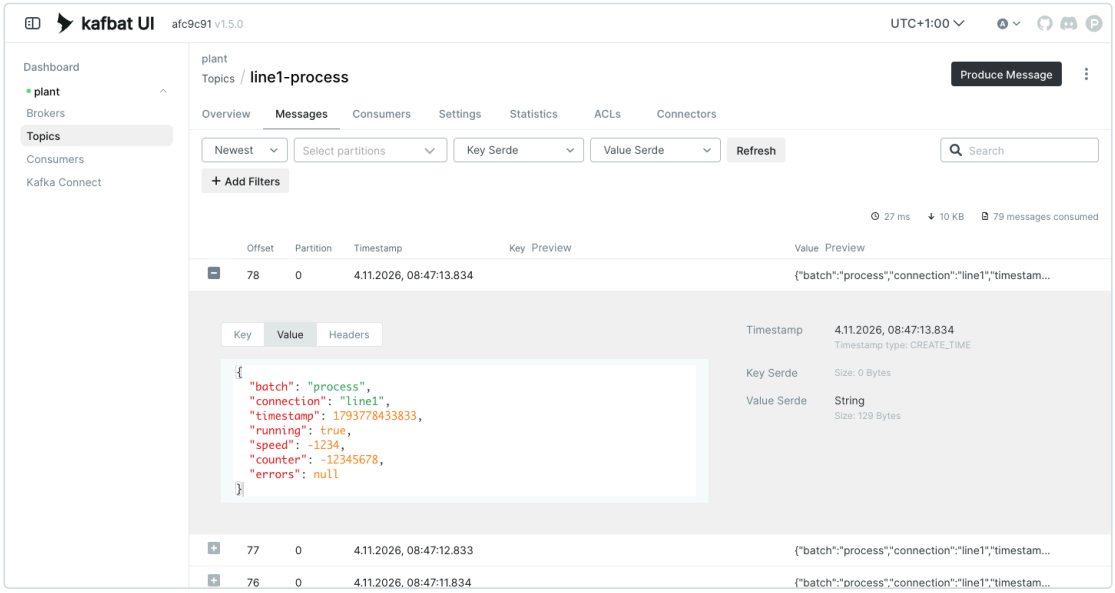
Der Datensatz hat **keinen Schlüssel**; seine Source-Partition ist `{batch, connection}`, sein Offset `{timestamp}`.

Jeder Knoten des Data Layer meldet seinen eigenen Typ; der Connector bildet ihn ab:

Typ im Data Layer	Connect-Typ	
bool8	BOOLEAN	
int8	INT8	
uint8, int16	INT16	vorzeichenlos 8 Bit erweitert
uint16, int32	INT32	vorzeichenlos 16 Bit erweitert
uint32, int64	INT64	vorzeichenlos 32 Bit erweitert
uint64	Decimal (Skala 0)	vorzeichenlos 64 Bit, exakt
float, float32	FLOAT32	
double, float64	FLOAT64	
string	STRING	
timestamp	Timestamp	
object (eine Struktur)	STRING	JSON
ein Array (arbool8, arint16, …)	ARRAY of STRING	jedes Element als Text

Der Data Layer hat **keine eigenen Zeittypen**: `TIME` und `TIME_OF_DAY` einer SPS sind uint32-Knoten und kommen als die Anzahl an, die sie enthalten (Millisekunden), ein `LTIME` ist ein uint64-Knoten und kommt in Nanosekunden an; `DATE` und `DATE_AND_TIME` sind timestamp-Knoten. Innerhalb einer Struktur kommen Datums- und Zeitwerte als die einfache Ganzzahl an, die die Steuerung hält; lesen Sie ein solches Element über seine eigene Adresse, um es typisiert zu erhalten.

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": "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}"
  }
}
```

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, sein Typ folgt der Typtabelle. Der Treiber schreibt eine Zahl im eigenen Typ des Knotens und weist eine Zahl, die der

Knoten nicht aufnehmen kann, oder einen Bruch für einen Ganzzahlknoten ab, bevor etwas gesendet wird. 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. Das Konto des Sinks braucht das Recht zu schreiben; ein Konto, das nur lesen darf, wird mit HTTP `403` abgewiesen.

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": "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 } }
  ]
}
```

Jede Verbindung braucht ihre Zugangsdaten (hier ein Konto für alle drei Pressen) und, bei werkseitigen Zertifikaten, den Fingerabdruck ihrer eigenen Steuerung. 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-ctrlx`, `…-s7`, …). **Verwenden Sie die Connector-Klassen des jeweiligen Treibers** - hier `…kafka.ctrlx.CtrlxSourceConnector`: 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.ctrlx.CtrlxSourceConnector` / `CtrlxSinkConnector` 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` lesen: eine Adresse, eine abgelehnte Anmeldung, ein Zertifikat, eine Lizenz
`errors` nennt `License validation failed`	keine Lizenz gilt	siehe Die Lizenz prüfen
das Zertifikat wird abgelehnt; das Worker-Log nennt einen `SHA-256 fingerprint`	das selbstsignierte Zertifikat der Steuerung	den Fingerabdruck hinterlegen (siehe Das Zertifikat der Steuerung)

Symptom	Ursache	was zu tun ist
ab und zu `TLS pin check failed: peer certificate SHA-256 fingerprint <Wert> is not in the pinned list`	die Steuerung zeigt mehrere Zertifikate vor, und nicht alle sind hinterlegt	den Wert in `tls.pinned-fingerprints` ergänzen, durch Kommas getrennt
`Driver 'ctrlx' requires a credential and none was supplied`	die Verbindung hat kein `credential`	eines mit `username-password` ergänzen (siehe Zugangsdaten)
HTTP `401 Unauthorized`	falscher Benutzername oder falsches Passwort	das Konto und die Werte prüfen, die `secrets.<name>` liefert
HTTP `403 Forbidden`	das Konto ist angemeldet, aber ihm fehlt die Berechtigung: Lesen braucht *Data Layer Global → View resources*, Schreiben eine Berechtigung zum Schreiben	die Rolle des Kontos in der Benutzerverwaltung der Steuerung anpassen
HTTP `404 Not Found` für eine Variable	den Knotenpfad gibt es nicht	den Pfad prüfen; SPS-Variablen liegen unter `plc/app/<Applikation>/sym/<Programm>/`
`INVALID_ADDRESS` für eine Variable	Leerzeichen oder ein leeres Segment in der Adresse	die Adresse korrigieren; die übrigen Variablen des Batches werden gelesen
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`).

Versionshistorie

1.3.0 — 2026-11-13

Basiert auf ToddySoft Connect 1.3.0.

- **Neu:** der ctrlX-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.