ToddySoft Connect ADS - Apache Kafka Connect

Ein Kafka-Connect-**Source- und Sink-Connector** für Beckhoff-TwinCAT-Steuerungen auf Basis des ADS-Treibers von ToddySoft Connect. Er liest Variablen aus Steuerungen mit TwinCAT 2 und TwinCAT 3 über ADS/AMS - unverschlüsselt oder als Secure ADS (TLS mit Client-Zertifikat oder Pre-Shared Key) - und veröffentlicht jeden Lesevorgang als einen Kafka-Datensatz; der Sink schreibt Felder von Datensätzen in Variablen.

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

	dieser Connector	Connector von Apache PLC4X 1.0.0
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 48898; 8016 für Secure ADS).
- **In der Steuerung**: eine **AMS-Route** für die Quell-AMS-Net-ID jedes Workers, vorab angelegt - der Connector legt keine Routen an (siehe Die Quell-AMS-Net-ID und die Route). Für symbolische Adressen muss das SPS-Programm laufen.
- Eine **ToddySoft-Lizenz für ADS** 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-ads-<version>.zip` - im Komponentenarchiv-Format von Confluent Hub, für Confluents Installationsprogramm;
- `toddysoft-kafka-connect-ads-<version>-plugin.zip` - ein einfaches Plugin-Verzeichnis.

Mit dem Confluent CLI (Confluent Platform)

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

Ein Worker im Container muss in jeder Verbindungs-URL seine **Quell-AMS-Net-ID** angeben: Die Vorgabe, abgeleitet aus der eigenen Adresse des Containers, ist keine, für die die Steuerung eine Route hat (siehe Die Quell-AMS-Net-ID und die Route).

Die Installation prüfen

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

listet `com.toddysoft.connect.integrations.kafka.ads.AdsSourceConnector` und `…AdsSinkConnector` 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/ads.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 [ads]`:

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

der Bericht sagt	Bedeutung	was zu tun ist
`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 ADS-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": "ads:tcp://192.168.10.20?source-ams-net-
        id=192.168.10.5.1.1&target-ams-port=851",
      "tags": {
        "temperature": "MAIN.fTemperature",
        "counter":     "MAIN.nCounter",
        "running":     "GVL.bRunning",
        "setpoint":    { "address": "MAIN.fSetpoint", "access":
        "write" }
      }
    }
  ],
  "batches": [
    { "id": "process", "connectionId": "line1", "tags":
        ["temperature", "counter"],
      "trigger": { "type": "timer", "intervalMillis": 1000 } },
    { "id": "state", "connectionId": "line1", "tags": ["running",
        "counter"],
      "trigger": { "type": "change", "watch": ["running"],
        "pollIntervalMillis": 200 } }
  ]
}
```

- **Verbindungs-URL**: `ads:tcp://<host>` (Port 48898) für unverschlüsseltes ADS, `ads:tls://<host>` oder `ads:tls-psk://<host>` (Port 8016) für Secure ADS. Parameter:
  - `target-ams-port` - die SPS-Laufzeit: `851` für TwinCAT 3 (die Vorgabe), `801` für TwinCAT 2;
  - `target-ams-net-id` - die AMS Net ID der Steuerung, Vorgabe `<ihre IP-Adresse>.1.1`; setzen Sie sie, wenn die Net ID der Steuerung dieser Konvention nicht folgt;
  - `source-ams-net-id` - die AMS Net ID dieses Clients, wie die Route der Steuerung sie nennt (siehe unten).
- **Adressen**, symbolisch, wie TwinCAT sie zeigt: `MAIN.fTemperature`, `GVL.sStatus`, ein Strukturelement `MAIN.stMotor.fSpeed`, ein Array-Element `MAIN.aValues[0]`, `MAIN.stArray[1].nField` oder `MAIN.matrix[2][3]`, ein ganzes Array `MAIN.aValues` (als eine Liste gelesen) oder eine ganze Struktur `MAIN.stMotor`. TwinCAT 2 meldet seine Symbolnamen in Großbuchstaben (`MAIN.BSTART`). Über Indexgruppe und Offset: `0x4020/0:DINT`, `0x4020/0[0..9]:DINT` (zehn DINTs), `0x4020/2.3:BIT`, `0x4020/0:STRING(80)`, `0x4020/0:WSTRING(40)`.

- **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 über ADS-Device-Notifications und reicht jede Änderung weiter).
- **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.

Die Quell-AMS-Net-ID und die Route

Eine TwinCAT-Steuerung antwortet nur einem Client, für den sie eine **AMS-Route** hat, und sie erkennt den Client an seiner **Quell-AMS-Net-ID** (*Source AMS Net ID*). Legen Sie die Route auf der Steuerung an, bevor sich der Connector verbindet - in TwinCAT XAE / System Manager (*Routes → Add*) oder in `StaticRoutes.xml` -, für die Net ID, die die Verbindung nennt, und die Adresse des Worker-Rechners. Der Connector registriert keine Routen. **TwinCAT 2 braucht für eine neue Route bis zu etwa zwei Minuten**, während es seinen AMS-Router neu startet; TwinCAT 3 übernimmt eine Route sofort.

Ohne `source-ams-net-id` nennt der Treiber `<die eigene Adresse des Workers>.1.1`. Setzen Sie sie ausdrücklich, wenn das nicht die Net ID ist, für die die Route angelegt wurde:

- **Der Worker läuft im Container.** Die eigene Adresse des Containers ist nicht die, die die Steuerung sieht; die Steuerung sieht den Rechner, auf dem der Container läuft. Nennen Sie die Net ID, für die die Route angelegt wurde, meist die Adresse dieses Rechners gefolgt von `.1.1`. Gemessen mit einem Worker `apache/kafka:4.4.0` in Docker an einer TwinCAT 3 (CX9240):

  ```
  ads:tcp://<plc>?source-ams-net-id=<die Adresse, für die die Route angelegt wurde>.1.1&target-ams-port=851
  ```

  Die Vorgabe, abgeleitet aus der Adresse des Containers, ist eine Net ID ohne Route, und die Steuerung antwortet ihr nicht.

- **Der Worker-Rechner hat mehrere Adressen**, zum Beispiel zwei im Subnetz der Steuerung: Die Vorgabe kann aus einer anderen abgeleitet sein als der, die die Route nennt. Setzen Sie `source-ams-net-id`, und binden Sie bei einem Worker, der direkt auf dem Rechner läuft, die Verbindung mit `tcp.local-address` (`tls.local-address` / `tls-psk.local-address` für Secure ADS) an die Adresse, für die die Route angelegt wurde.

- **Der Worker läuft auf dem TwinCAT-Rechner selbst.** Beide Vorgaben wären die eigene Net ID der SPS: Wählen Sie eine andere, zum Beispiel `<Adresse>.1.2`, und legen Sie dafür eine Route an, die auf die eigene Adresse des Rechners zeigt.

TwinCAT hält eine Verbindung je Quell-AMS-Net-ID. Ein zweiter Client mit derselben Net ID - ein weiterer Worker, ein Ignition-Gateway, ein anderes ToddySoft-Produkt auf demselben Rechner - nimmt dem ersten die Verbindung weg, und beide verdrängen sich immer wieder (`Connection closed by remote` Millisekunden nach dem Verbinden). Geben Sie jedem Worker und jedem anderen Client eine eigene Net ID und Route. Innerhalb eines Plugins teilen sich Connectoren mit derselben Verbindung eine Verbindung (siehe Mehrere Treiber-Plugins auf einem Worker); zwei Verbindungen mit verschiedenen URLs zur selben Steuerung und mit derselben Net ID konkurrieren trotzdem.

Secure ADS

Secure ADS (TwinCAT 3.1 ab Build 4024) ist TLS auf Port 8016, mit einer von zwei Arten von Zugangsdaten (siehe Zugangsdaten):

- `ads:tls-psk://` - ein **Pre-Shared Key** mit Identität, wie in der Route der Steuerung eingerichtet. Keine Zertifikate. TwinCAT leitet den Schlüssel als SHA-256 der Identität (in Großbuchstaben) gefolgt vom Passwort ab: `echo -n "KAFKA1MySecret" | shasum -a 256` ergibt die 64 Hex-Zeichen, die der Connector braucht.
- `ads:tls://` - das **Zertifikat** dieses Clients aus einem PKCS12-Keystore. Die Steuerung braucht eine Route für dieses Zertifikat. Gibt es die Keystore-Datei noch nicht, legt die erste Verbindung sie an (ein selbstsigniertes Zertifikat, nur für den Eigentümer lesbar, geschützt mit dem Keystore-Passwort der Zugangsdaten) und wird mit `UnknownCert` abgelehnt, unter Nennung der Datei, ihres Common Name und Fingerabdrucks: Legen Sie die Route dafür an und starten Sie den Connector neu. Legen Sie die Datei auf einen dauerhaften, beschreibbaren Pfad: Ein neuer Schlüssel braucht eine neue Route.

`machine-name` ist der Name, der in der Secure-ADS-Sitzung genannt wird, standardmäßig der Hostname ohne Domäne (im Container: der Hostname des Containers). Bei einem selbstsignierten Client-Zertifikat muss er dem Common Name des Zertifikats entsprechen, sonst antwortet die Steuerung mit `CnMismatch`.

Das Zertifikat der Steuerung wird geprüft (`tls.verify`, standardmäßig an). Ein Zertifikat, das Ihre eigene CA signiert hat, braucht auf jedem Worker einen Truststore (`tls.trust-store=/etc/kafka-connect/plc-ca.p12&tls.trust-store-password=…`). Ein selbstsigniertes Zertifikat der Steuerung lässt sich über seinen SHA-256-Fingerabdruck hinterlegen, den das Log des Workers nennt, wenn das Zertifikat abgelehnt wird:

```
ads:tls://192.168.10.20?source-ams-net-
id=192.168.10.5.1.1&tls.pinned-fingerprints=95:62:7b:…:d1
```

`tls.verify=false` akzeptiert jedes Zertifikat und schützt nicht vor dem Abhören.

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.Fi
```

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

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

Eine Verbindung über Secure ADS mit Pre-Shared Key:

```
{
  "id": "line1",
  "url": "ads:tls-psk://192.168.10.20?source-ams-net-
        id=192.168.10.5.1.1&target-ams-port=851",
  "credential": { "method": "pre-shared-key", "pskIdentity":
        "KAFKA1", "pskKey": "{{line1.psk}}" },
  "tags": { "temperature": "MAIN.fTemperature" }
}
```

die Verbindung	`method`	anzugeben
unverschlüsseltes ADS (`ads:tcp://`)	kein `credential`	nichts
Secure ADS mit Pre-Shared Key (`ads:tls-psk://`)	`pre-shared-key`	`pskIdentity`, `pskKey` (hexadezimal)
Secure ADS mit Client-Zertifikat (`ads:tls://`)	`certificate`	`keystore` (ein Pfad auf jedem Worker), `keystorePassword`

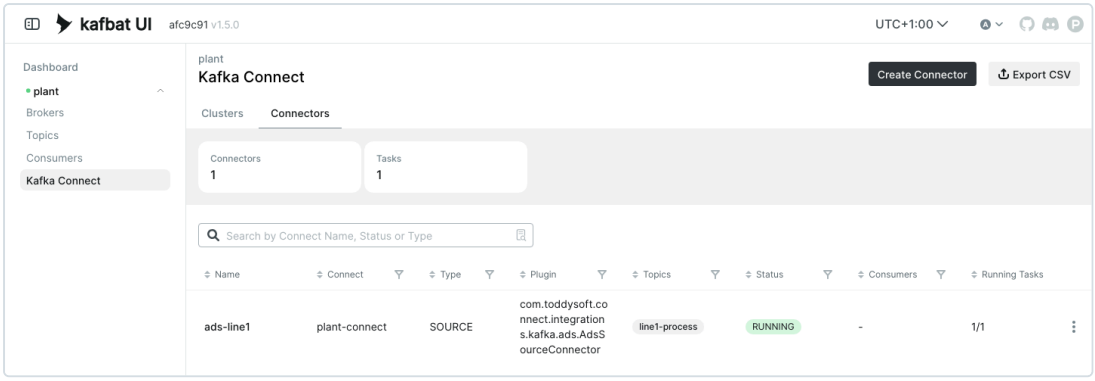
Kafka protokolliert die Konfigurationswerte dieser Connectoren nie, und der Worker schreibt den Platzhalter `${file:…}`, nicht den Wert, in sein Config-Topic. Beschränken Sie das Config-Topic trotzdem mit ACLs.

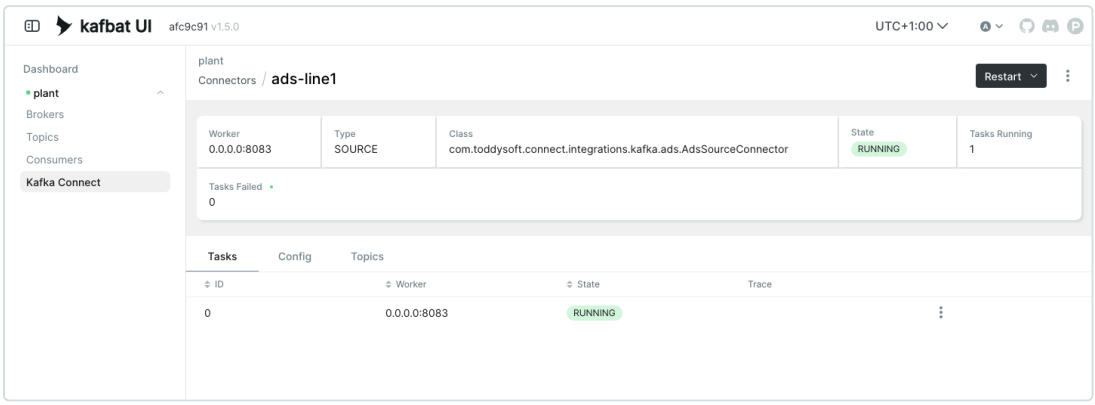
Der Source-Connector

```
{
  "name": "ads-line1",
  "config": {
    "connector.class":
        "com.toddysoft.connect.integrations.kafka.ads.AdsSourceConnector",
    "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 @ads-line1.json http://localhost:8083/connectors` legt ihn an. Eine Verbindung mit Zugangsdaten braucht hier zusätzlich ihr `secrets.<name>` (siehe Zugangsdaten).

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 TwinCAT 3 (CX9240), die die Variablen `MAIN.g_b1`, `MAIN.g_s16` und `MAIN.g_s32` einmal pro Sekunde liest.

Datensätze

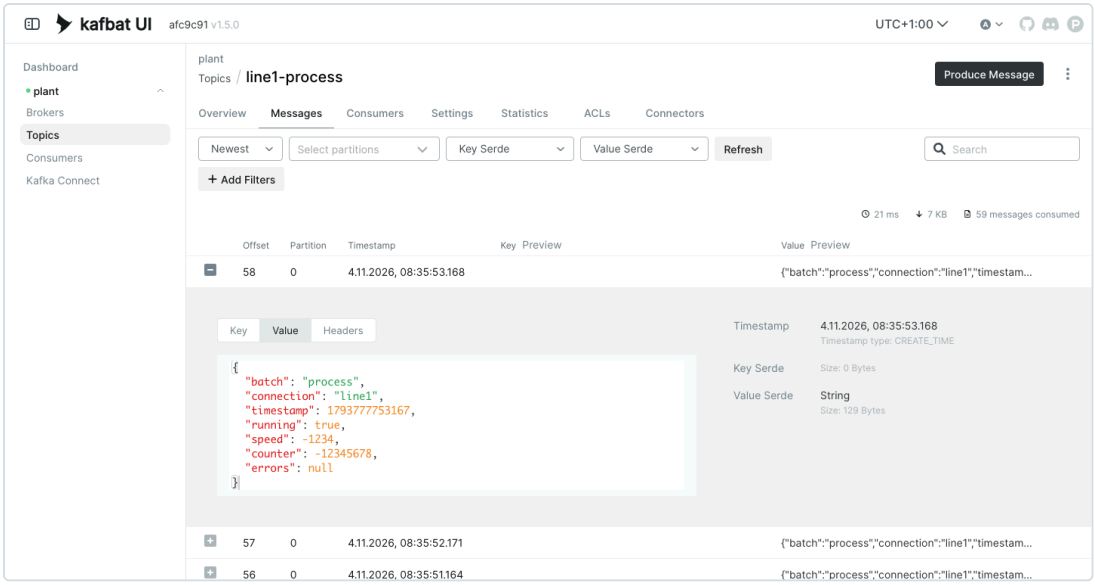
Ein Datensatz je Lesevorgang. Sein Wert ist ein `Struct`:

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}`.

Typ in der Steuerung	Connect-Typ	
BOOL, BIT	BOOLEAN	BIT: ein Bit-Element einer Struktur
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	
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	
eine Struktur oder ein Funktionsbaustein	STRING	JSON
ein Array	ARRAY of STRING	jedes Element als Text, eine Struktur als JSON

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": "ads-line1-setpoints",
  "config": {
    "connector.class":
        "com.toddysoft.connect.integrations.kafka.ads.AdsSinkConnector",
    "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, sein Typ folgt der Typtabelle. 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": "MAIN.fSpeed", "count":
        "MAIN.nCount" } },
  "connections": [
    { "id": "press-1", "url": "ads:tcp://192.168.10.31?source-ams-
        net-id=192.168.10.5.1.1", "tagSets": ["press"] },
    { "id": "press-2", "url": "ads:tcp://192.168.10.32?source-ams-
        net-id=192.168.10.5.1.1", "tagSets": ["press"] },
    { "id": "press-3", "url": "ads:tcp://192.168.10.33?source-ams-
        net-id=192.168.10.5.1.1", "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. Eine Quell-AMS-Net-ID zu mehreren Steuerungen ist kein Problem: TwinCAT hält auf jeder Steuerung eine Verbindung je Net ID.

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. Jede Steuerung braucht deshalb eine Route für jeden Worker, der ihre Task ausführen kann. 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-ads`, `…-s7`, …). **Verwenden Sie die Connector-Klassen des jeweiligen Treibers** - hier `…kafka.ads.AdsSourceConnector`: 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 - bei ADS heißt das auch, dass sie sich nicht gegenseitig verdrängen.

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.ads.AdsSourceConnector` / `AdsSinkConnector` 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 fehlende Route, ein Zertifikat, eine Lizenz
`errors` nennt `License validation failed`	keine Lizenz gilt	siehe Die Lizenz prüfen
TwinCAT 3: jede Verbindung läuft in eine Zeitüberschreitung; TwinCAT 2: `Transport is closed`	die Steuerung hat keine Route für die Quell-AMS-Net-ID	`source-ams-net-id` prüfen (im Container immer setzen) und die Route anlegen, siehe Die Quell-AMS-Net-ID und die Route
TwinCAT 2: die Route wurde gerade angelegt, und die Verbindung schlägt trotzdem fehl	die Steuerung startet ihren AMS-Router neu, bis zu etwa zwei Minuten	warten und nach zwei Minuten erneut nachsehen

Symptom	Ursache	was zu tun ist
`Connection closed by remote` Millisekunden nach dem Verbinden, Verbindungen brechen immer wieder ab	ein anderer Client verwendet zu dieser Steuerung dieselbe Quell-AMS-Net-ID	jedem Worker und jedem anderen Client eine eigene Net ID und Route geben
eine Variable `null` mit einem Nicht-gefunden-Grund in `errors`	das Symbol gibt es im laufenden Programm nicht, oder das Programm läuft nicht	den Namen aus TwinCAT übernehmen (TwinCAT 2: in Großbuchstaben), die SPS in RUN setzen
`UnknownCert`	die Steuerung hat keine Route für das Zertifikat dieses Clients (ein gerade angelegter oder ersetzter Keystore)	die Route für das Zertifikat anlegen, siehe Secure ADS
`CnMismatch`	der genannte Rechnername weicht vom Common Name des Client-Zertifikats ab	`machine-name` auf den Common Name setzen
der TLS-Handshake lehnt das Zertifikat der Steuerung ab	nicht von einer CA signiert, der der Worker vertraut	einen Truststore nennen oder den Fingerabdruck aus dem Log hinterlegen (siehe Secure ADS)
Verbindungen über `ads:tls-psk` werden abgelehnt	Identität oder Schlüssel weichen von der Route der Steuerung ab	den Schlüssel aus der Identität in Großbuchstaben und dem Passwort neu berechnen

Symptom	Ursache	was zu tun ist
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 ADS-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.