

Release G: Kafka Connect

- [Introduction](#)
- [Connect Demo](#)
 - [Postgres JDBC Sink Connector \(Confluent\)](#)
 - [Influxdb Sink Connector \(Apache Camel\)](#)
 - [Minio Sink Connector \(Apache Camel\)](#)
 - [File Sink Connector \(Apache Camel\)](#)
- [Links](#)

Introduction

Kafka Connect allows you to continuously ingest data from external systems into Kafka via connect sources and write data from Kafka to external system via connect sinks.

Various plugins are available for a variety of different data sources and data sinks.

[Confluent Plugins](#)

[Camel Plugins](#)

[Debezium Plugins](#)

Single Message *Transformations* (SMTs) are applied to messages as they flow through *Connect*.

Connect Demo

Postgres JDBC Sink Connector (Confluent)

Prepare the connector image

Download the JDBC sink connector from [JDBC Connector \(Source and Sink\)](#)

Create a docker file for this connector

Docker
<pre>FROM quay.io/strimzi/kafka:0.32.0-kafka-3.3.1 USER root:root RUN mkdir -p /opt/kafka/plugins/jdbc COPY ./confluentinc-kafka-connect-jdbc-10.6.0.zip /opt/kafka/plugins/j dbc/ RUN unzip /opt/kafka/plugins/jdbc/confluentinc-kafka-connect-jdbc-10.6.0.zip -d /opt/kafka/plugins/jdbc RUN rm /opt/kafka/plugins/jdbc/confluentinc-kafka-connect-jdbc-10.6.0.zip USER 1001</pre>

Build the image and copy it to your docker repository

docker build -f Dockerfile . t ktimoney/strimzi connector

Prepare the postges database

Setup a new username/password and schema for Kafka connect to write to

Postgres

```
SELECT 'CREATE DATABASE kafka'
WHERE NOT EXISTS (SELECT FROM pg_database WHERE datname = 'kafka')\gexec
DO $$
BEGIN
    IF NOT EXISTS (SELECT FROM pg_user WHERE username = 'kafka') THEN
        CREATE USER kafka WITH PASSWORD 'kafka';
        GRANT ALL PRIVILEGES ON DATABASE kafka TO kafka;
    END IF;
END
$$;
```

Prepare the message producer

In this example we are going to deserialize the kafka key and value to json, for this to work we need to include the json schema with the message.

They will take the following format:

Schema & Payload

```
{
  "schema": {
    "type": "struct",
    "optional": false,
    "version": 1,
    "fields": [
      {
        "field": "id",
        "type": "string",
        "optional": true
      }
    ]
  },
  "payload": {
    "id": 1
  }
}
```

The above structure can be represented in go lang using the following structs:

Go Structs

```
type SchemaPayload[T any] struct {
    Schema Schema `json:"schema"`
    Payload T `json:"payload"`
}

type Schema struct {
    Type string `json:"type"`
    Optional bool `json:"optional"`
    Version int `json:"version"`
    Fields []Fields `json:"fields"`
}

type Fields struct {
    Field string `json:"field"`
    Type string `json:"type"`
    Optional bool `json:"optional"`
}

type KeyPayload struct {
    Id string `json:"id"`
}

type ValuePayload struct {
    Text string `json:"text"`
}
```

We can use the same SchemaPayload struct for both keys and values, we only need to provide the Payload struct type we initialize it with.

[Create the KafkaConnect object](#)

KafkaConnect

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: connect
  namespace: kafka
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
---
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  namespace: kafka
  annotations:
    # use-connector-resources configures this KafkaConnect
    # to use KafkaConnector resources to avoid
    # needing to call the Connect REST API directly
    strimzi.io/use-connector-resources: "true"
spec:
  image: ktimoney/strimzi-connect
  replicas: 1
  bootstrapServers: my-cluster-kafka-bootstrap:9093
  tls:
    trustedCertificates:
      - secretName: my-cluster-cluster-ca-cert
        certificate: ca.crt
      - secretName: my-cluster-clients-ca-cert
        certificate: ca.crt
  authentication:
    type: tls
    certificateAndKey:
      secretName: connect
      certificate: user.crt
      key: user.key
  config:
    group.id: connect-cluster
    offset.storage.topic: connect-cluster-offsets
    config.storage.topic: connect-cluster-configs
    status.storage.topic: connect-cluster-status
    # -1 means it will use the default replication factor configured in the broker
    config.storage.replication.factor: -1
    offset.storage.replication.factor: -1
    status.storage.replication.factor: -1
```

We are connecting to the 9093 tls port so we include required certificates in our configuration.

The image tag references the image we built earlier : ktimoney/strimzi-connect

[Create the Sink connector](#)

KafkaConnector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-sink-connector
  namespace: kafka
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: io.confluent.connect.jdbc.JdbcSinkConnector
  tasksMax: 2
  config:
    topics: my-topic
    connection.url: jdbc:postgresql://postgres.default:5432/kafka
    connection.user: kafka
    connection.password: kafka
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: true
    value.converter.schemas.enable: true
    pk.mode: record_key
    pk.fields: id
    auto.create: true
    delete.enabled: true
```

The KafkaConnector is configured to use the "io.confluent.connect.jdbc.JdbcSinkConnector" class.

The converters are set to "org.apache.kafka.connect.json.JsonConverter"

The pk.mode is set to "record_key" and pk.fields is set to "id", this means it will use the id field from the key as the primary key.

The database connection details are also included.

Influxdb Sink Connector (Apache Camel)

Prepare the connector image

Download the Influxdb connector: wget <https://repo.maven.apache.org/maven2/org/apache/camel/kafkaconnector/camel-influxdb-kafka-connector/0.8.0/camel-influxdb-kafka-connector-0.8.0-package.tar.gz>

Create a docker file for this connector

Docker

```
FROM quay.io/strimzi/kafka:0.22.1-kafka-2.7.0
USER root:root
RUN mkdir -p /opt/kafka/plugins/camel
COPY ./camel-influxdb-kafka-connector-0.8.0-package.tar.gz /opt/kafka/plugins/camel/
RUN tar -xvzf /opt/kafka/plugins/camel/camel-influxdb-kafka-connector-0.8.0-package.tar.gz --directory /opt/kafka/plugins/camel
RUN rm /opt/kafka/plugins/camel/camel-influxdb-kafka-connector-0.8.0-package.tar.gz
USER 1001
```

Build the image and copy it to your docker repository

```
docker build -f Dockerfile . t ktimoney/strimzi_influxdb_connector
```

Prepare the message producer

In this example we are going to deserialize the value to json, we can do this without using a schema.

They will take the following format:

Schema & Payload

```
{
  "camelInfluxDB.MeasurementName": "disk_space"
  "time": "2023-01-23T13:03:31Z"
  "host": "localhost"
  "region": "IE-region"
  "used": "19%"
  "free": "81%"
}
```

The important thing to note here is that one of the "keys" has to be "camelInfluxDB.MeasurementName", this is required by the camel transformer when converting into an influxdb point object.

The above structure can be represented in golang using the following struct:

Go Structs

```
type InfluxPayload struct {
    Measurement string `json:"camelInfluxDB.MeasurementName"`
    Time        string `json:"time"`
    Host        string `json:"host"`
    Region      string `json:"region"`
    Used        string `json:"used"`
    Free        string `json:"free"`
}
```

[Create the KafkaConnect object](#)

KafkaConnect

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: connect
  namespace: kafka
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
---
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  namespace: kafka
  annotations:
    # use-connector-resources configures this KafkaConnect
    # to use KafkaConnector resources to avoid
    # needing to call the Connect REST API directly
    strimzi.io/use-connector-resources: "true"
spec:
  image: ktimoney/strimzi-influxdb-connect
  replicas: 1
  bootstrapServers: my-cluster-kafka-bootstrap:9093
  tls:
    trustedCertificates:
      - secretName: my-cluster-cluster-ca-cert
        certificate: ca.crt
      - secretName: my-cluster-clients-ca-cert
        certificate: ca.crt
  authentication:
    type: tls
    certificateAndKey:
      secretName: connect
      certificate: user.crt
      key: user.key
  config:
    group.id: connect-cluster
    offset.storage.topic: connect-cluster-offsets
    config.storage.topic: connect-cluster-configs
    status.storage.topic: connect-cluster-status
    config.storage.replication.factor: 1
    offset.storage.replication.factor: 1
    status.storage.replication.factor: 1
```

We are connecting to the 9093 tls port so we include required certificates in our configuration.

The image tag references the image we built earlier : ktimoney/strimzi-influx-connect

[Create the Sink connector](#)

KafkaConnector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-sink-connector
  namespace: kafka
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: org.apache.camel.kafkaconnector.influxdb.CamelInfluxdbSinkConnector
  tasksMax: 1
  config:
    topics: my-topic
    errors.deadletterqueue.topic.name: my-topic-dl
    errors.deadletterqueue.topic.replication.factor: 1
    key.converter: org.apache.kafka.connect.storage.StringConverter
    value.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: false
    value.converter.schemas.enable: false
    key.ignore: true
    auto.create: true
    camel.beans.influx: "#class:org.influxdb.InfluxDBFactory#connect('http://influxdb.default:8086',
'influxdb', 'influxdb')"
    camel.sink.path.connectionBean: influx
    camel.sink.endpoint.databaseName: ts_host_metrics
    camel.sink.endpoint.operation: insert
    camel.sink.endpoint.retentionPolicy: autogen
```

The `KafkaConnector` is configured to use the `"org.apache.camel.kafkaconnector.influxdb.CamelInfluxdbSinkConnector"` class.

The value converter is set to `"org.apache.kafka.connect.json.JsonConverter"`.

The camel connector comes with a in built `TypeConverter` called `CamelInfluxDbConverters` which will read the json produced as a `Map<String, Object>` map) and return an `Influxdb Point` object.

The database connection details are specified in `camel.beans.influx`.

This sets up a connection bean to be used by `camel.sink.path.connectionBean`.

Note: It is important to wrap the `camel.beans.influx` parameter in quotes otherwise it will be treated as a comment.

The yaml for the influxdb pod is available here: [influxdb.yaml](#)

When the connector is running it will produce records to influxdb like the following:

InfluxDB query

```
{'pretty': 'true', 'db': 'ts_host_metrics', 'q': 'SELECT "region", "host", "free", "used" FROM "disk_space"
WHERE "host"=\'localhost\''}
{
  "results": [
    {
      "statement_id": 0,
      "series": [
        {
          "name": "disk_space",
          "columns": [
            "time",
            "region",
            "host",
            "free",
            "used"
          ],
          "values": [
            [
              "2023-01-23T13:03:32.1246436Z",
              "IE-region",
              "localhost",
              "81%",
              "19%"
            ]
          ]
        }
      ]
    }
  ]
}
```

Minio Sink Connector (Apache Camel)

Prepare the connector image

Download the minio sink connector: wget <https://repo.maven.apache.org/maven2/org/apache/camel/kafkaconnector/camel-file-kafka-connector/3.20.0/camel-file-kafka-connector-3.20.0-package.tar.gz>

We need to create some custom classes:

CamelMinioConverters.java - this will allow us to deserialize to json and convert the hashmap to an inputstream

CamelMinioConverters

```
package com.custom.convert.minio;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.util.Map;
import org.apache.camel.Converter;

@Converter(generateLoader = true)
public final class CamelMinioConverters {

    private CamelMinioConverters() {
    }

    @SuppressWarnings("deprecation")
    @Converter
    public static InputStream fromMapToInputStream(Map<String, Object> map) {

        String json = "{}";
        try {
            json = new ObjectMapper().writeValueAsString(map);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
        return new ByteArrayInputStream(json.getBytes());
    }
}
```

StringAggregator.java - this will allow us to aggregate the messages and set the file name

StringAggregator

```
package com.custom.aggregate.minio;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import org.apache.camel.AggregationStrategy;
import org.apache.camel.Exchange;
import org.apache.camel.Message;

public class CustomStringAggregator implements AggregationStrategy {

    //Override
    public Exchange aggregate(Exchange oldExchange, Exchange newExchange) {
        Message newIn = newExchange.getIn();
        Map<String, String> keyMap = (HashMap) newIn.getHeaders().get("camel.kafka.connector.record.key");
        String key = keyMap.get("id");
        SimpleDateFormat sdf = new SimpleDateFormat("ddMMyy-hhmmss-SSS");
        String fileName = "ExchangeKafka-" + key + "-" + sdf.format( new Date() ) + ".json";
        newIn.setHeader("file", fileName);

        // lets append the old body to the new body
        if (oldExchange == null) {
            return newExchange;
        }

        String body = oldExchange.getIn().getBody(String.class);
        if (body != null) {
            String newBody = newIn.getBody(String.class);
            if (newBody != null) {
                body += System.lineSeparator() + newBody;
            }

            newIn.setBody(body);
        }
        return newExchange;
    }
}
```

We can then create a custom jar for our code and copy it into the docker container.

Note: You can also set the filename in the Kafka header using either CamelHeader.file or CamelHeader.ce-file

Create a docker file for this connector

Docker

```
FROM quay.io/strimzi/kafka:0.32.0-kafka-3.3.1
USER root:root
RUN mkdir -p /opt/kafka/plugins/camel
COPY ./camel-minio-sink-kafka-connector-3.20.0-package.tar.gz /opt/kafka/plugins/camel/
RUN tar -xvzf /opt/kafka/plugins/camel/camel-minio-sink-kafka-connector-3.20.0-package.tar.gz --directory /opt/kafka/plugins/camel
COPY ./custom-converter-1.0.0.jar /opt/kafka/plugins/camel/camel-minio-sink-kafka-connector/
RUN rm /opt/kafka/plugins/camel/camel-minio-sink-kafka-connector-3.20.0-package.tar.gz
RUN rm -rf /opt/kafka/plugins/camel/docs
USER 1001
```

Build the image and copy it to your docker repository

```
docker build -f Dockerfile . t ktimoney/strimzi_minio_connector
```

Prepare the message producer

In this example we are going to deserialize the value to a string. (Same producer as the influxdb example)

They will take the following format:

Schema & Payload

```
{
  "camelInfluxDB.MeasurementName": "disk_space"
  "time": "2023-01-23T13:03:31Z"
  "host": "localhost"
  "region": "IE-region"
  "used": "19%"
  "free": "81%"
}
```

With the type converter in place we can also deserialize as a json object.

They key will be a json that looks like this: {"id": "119"}

Create the KafkaConnect object

KafkaConnect

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: connect
  namespace: kafka
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
---
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  namespace: kafka
  annotations:
    # use-connector-resources configures this KafkaConnect
    # to use KafkaConnector resources to avoid
    # needing to call the Connect REST API directly
    strimzi.io/use-connector-resources: "true"
spec:
  image: ktimoney/strimzi-minio-connect
  replicas: 1
  bootstrapServers: my-cluster-kafka-bootstrap:9093
  tls:
    trustedCertificates:
      - secretName: my-cluster-cluster-ca-cert
        certificate: ca.crt
      - secretName: my-cluster-clients-ca-cert
        certificate: ca.crt
  authentication:
    type: tls
    certificateAndKey:
      secretName: connect
      certificate: user.crt
      key: user.key
  config:
    group.id: connect-cluster
    offset.storage.topic: connect-cluster-offsets
    config.storage.topic: connect-cluster-configs
    status.storage.topic: connect-cluster-status
    # -1 means it will use the default replication factor configured in the broker
    config.storage.replication.factor: -1
    offset.storage.replication.factor: -1
    status.storage.replication.factor: -1
```

We are connecting to the 9093 tls port so we include required certificates in our configuration.

The image tag references the image we built earlier : ktimoney/strimzi-minio-connect

[Create the Sink connector](#)

KafkaConnector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-sink-connector
  namespace: kafka
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: org.apache.camel.kafkaconnector.miniosink.CamelMiniosinkSinkConnector
  tasksMax: 1
  config:
    topics: my-topic
    key.converter: org.apache.kafka.connect.json.JsonConverter
    value.converter: org.apache.kafka.connect.storage.StringConverter
    key.converter.schemas.enable: false
    value.converter.schemas.enable: false
    camel.beans.aggregate: "#class:com.custom.aggregate.minio.CustomStringAggregator"
    camel.aggregation.size: 1
    camel.aggregation.timeout: 20000
    camel.aggregation.disable: false
    camel.kamelet.minio-sink.bucketName: camel
    camel.kamelet.minio-sink.accessKey: ybK7RleFUKdDeYBf
    camel.kamelet.minio-sink.secretKey: X0Y4zK84bWdefRTlJrnPz0b1l0A60Qj2
    camel.kamelet.minio-sink.endpoint: http://minio.default:9000
    camel.kamelet.minio-sink.autoCreateBucket: true
```

The KafkaConnector is configured to use the "org.apache.camel.kafkaconnector.miniosink.CamelMiniosinkSinkConnector class.

The value converter is set to "org.apache.kafka.connect.storage.StringConverter".

The minio connection details are specified in camel.kamelet.minio-sink parameters.

The aggregation size is set to 1 so it won't aggregate any of the files

This timeout is also specified so it will only aggregate records arriving within this time out value (20 seconds).

These values can be adjusted depending on your requirements.

The main purpose of the aggregator in this example is for setting the file name, this is done by setting a header with a key of "file".

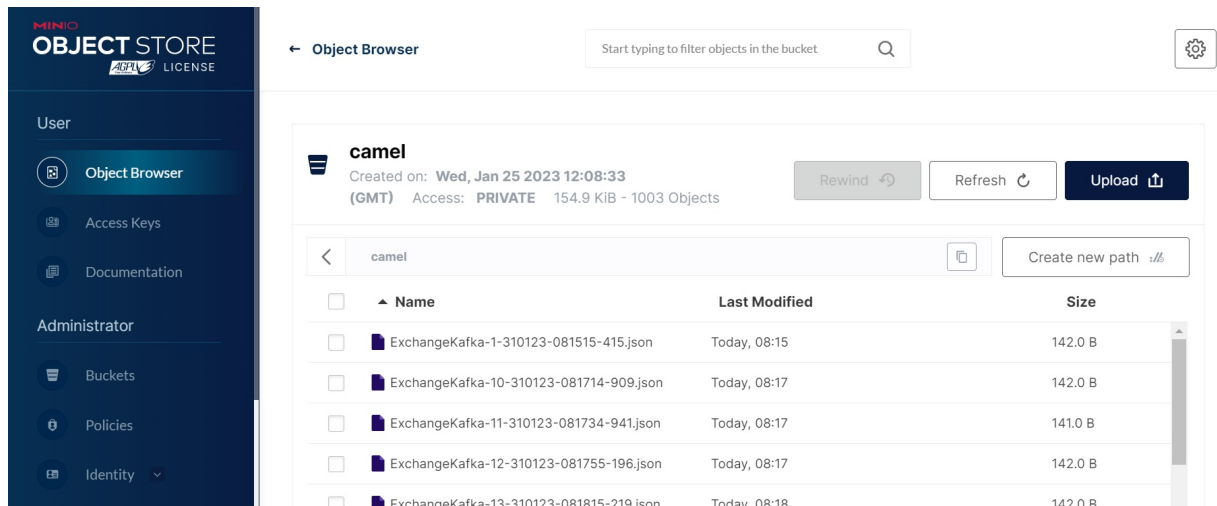
The yaml for the minio pod is available here: [minio.yaml](#)

When the connector is running it copies the kafka records into minio as files:

The screenshot displays the Minio Object Store web interface. On the left is a dark sidebar with navigation links: User, Object Browser (selected), Access Keys, Documentation, Administrator, Buckets, Policies, and Identity. The main area shows the 'Object Browser' for a bucket named 'camel'. At the top, it indicates the bucket was created on 'Wed, Jan 25 2023 12:08:33 (GMT)', has 'PRIVATE' access, and contains '12.5 KiB - 90 Objects'. Below this is a table of objects:

Name	Last Modified	Size
BF0A3477E3B874F-000000000000000000	Today, 12:09	142.0 B
BF0A3477E3B874F-000000000000000001	Today, 12:09	142.0 B
BF0A3477E3B874F-000000000000000002	Today, 12:09	142.0 B
BF0A3477E3B874F-000000000000000003	Today, 12:09	142.0 B
BF0A3477E3B874F-000000000000000004	Today, 12:09	142.0 B

Here's a screen shot where we use the aggregator to set the file name:



File Sink Connector (Apache Camel)

Prepare the connector image

Download the file connector: wget <https://repo.maven.apache.org/maven2/org/apache/camel/kafkaconnector/camel-file-kafka-connector/3.20.0/camel-file-kafka-connector-3.20.0-package.tar.gz>

Create a docker file for this connector

Docker

```
FROM quay.io/strimzi/kafka:0.32.0-kafka-3.3.1
USER root:root
RUN mkdir -p /opt/kafka/plugins/camel
COPY ./camel-file-kafka-connector-3.20.0-package.tar.gz /opt/kafka/plugins/camel/
RUN tar -xvzf /opt/kafka/plugins/camel/camel-file-kafka-connector-3.20.0-package.tar.gz --directory /opt/kafka/plugins/camel
RUN rm /opt/kafka/plugins/camel/camel-file-kafka-connector-3.20.0-package.tar.gz
RUN rm -rf /opt/kafka/plugins/camel/docs
USER 1001
```

Build the image and copy it to your docker repository

```
docker build -f Dockerfile . -t ktimoney/strimzi_file_connector
```

Prepare the message producer

In this example we are going to deserialize the value to a string.

The value will take the following format:

Schema & Payload

```
{
  "camelInfluxDB.MeasurementName": "disk_space"
  "time": "2023-01-23T13:03:31Z"
  "host": "localhost"
  "region": "IE-region"
  "used": "19%"
  "free": "81%"
}
```

Due to some limitation with Strimzi KafkaConnect we will need to use a mutating webhook to attach a persistent volume to our connector pod.

The go code for this is available here: [kafka-webhook.go](https://github.com/ktimoney/kafka-webhook)

We need to create a docker image for this program

Webhook docker

```
FROM golang:latest
RUN mkdir /app
COPY ./kafka-webhook /app
RUN chmod +x /app/kafka-webhook
WORKDIR /app
ENTRYPOINT ["/app/kafka-webhook"]
```

```
docker build -f Dockerfile . -t ktimoney/kafka-webhook
```

Mutating webhooks require https so we also need to create some certificates.

Make sure cfssl is installed on you system, then run the following commands.

```
cfssl gencert -initca ./ca-csr.json | cfssljson -bare ca
```

```
cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -hostname="kafka-webhook,kafka-webhook.kafka.svc.cluster.local,kafka-webhook.kafka.svc,localhost,127.0.0.1" \
  -profile=default \
  ca-csr.json | cfssljson -bare webhook-cert
```

Run the following commands to create the values you need to use in the yaml file:

```
TLS_CRT=$(cat webhook-cert.pem | base64 | tr -d '\n')
```

```
TLS_KEY=$(cat webhook-cert-key.pem | base64 | tr -d '\n')
```

```
CA_BUNDLE=$(openssl base64 -A <"ca.pem")
```

Replace the values in : [kafka-webhook.yaml](#)

Note: You can also configure the hostPath and the containerPath as part of the deployment:

Container command args

```
containers:
- name: kafka-webhook
  image: ktimoney/kafka-webhook
  imagePullPolicy: IfNotPresent
  command: ["/app/kafka-webhook"]
  args: [
    "-port", "8443",
    "-tlsCertFile", "/certs/tls.crt",
    "-tlsKeyFile", "/certs/tls.key",
    "-hostPath", "/var/strimzi/files",
    "-containerPath", "/opt/kafka/data"
  ]
```

Run: `kubectl create -f kafka-webhook.yaml` to start the pod.

Note: You can delete the webhook once the connector pod starts.

[Create the KafkaConnect object](#)

KafkaConnect

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: connect
  namespace: kafka
  labels:
    strimzi.io/cluster: my-cluster
spec:
  authentication:
    type: tls
---
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnect
metadata:
  name: my-connect-cluster
  namespace: kafka
  labels:
    webhook: "true"
  annotations:
    # use-connector-resources configures this KafkaConnect
    # to use KafkaConnector resources to avoid
    # needing to call the Connect REST API directly
    strimzi.io/use-connector-resources: "true"
spec:
  image: ktimoney/strimzi-file-connect
  replicas: 1
  bootstrapServers: my-cluster-kafka-bootstrap:9093
  tls:
    trustedCertificates:
      - secretName: my-cluster-cluster-ca-cert
        certificate: ca.crt
      - secretName: my-cluster-clients-ca-cert
        certificate: ca.crt
  authentication:
    type: tls
    certificateAndKey:
      secretName: connect
      certificate: user.crt
      key: user.key
  config:
    group.id: connect-cluster
    offset.storage.topic: connect-cluster-offsets
    config.storage.topic: connect-cluster-configs
    status.storage.topic: connect-cluster-status
    config.storage.replication.factor: 1
    offset.storage.replication.factor: 1
    status.storage.replication.factor: 1
```

We are connecting to the 9093 tls port so we include required certificates in our configuration.

The image tag references the image we built earlier : ktimoney/strimzi-file-connect

In the metadata section of KafkaConnect I have added a new label **webhook: "true"**, this will be picked up by the MutatingWebhookConfiguration:

MutatingWebhookConfiguration

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: kafka-webhook-config
  namespace: kafka
  annotations:
    cert-manager.io/inject-ca-from-secret: kafka/kafka-webhook-cert
webhooks:
- name: kafka-webhook.kafka.svc.cluster.local
  admissionReviewVersions:
    - "v1"
  sideEffects: "None"
  timeoutSeconds: 30
  objectSelector:
    matchLabels:
      webhook: "true"
```

matchLabels: webhook: "true"

[Create the Sink connector](#)

KafkaConnector

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaConnector
metadata:
  name: my-sink-connector
  namespace: kafka
  labels:
    strimzi.io/cluster: my-connect-cluster
spec:
  class: org.apache.camel.kafkaconnector.file.CamelFileSinkConnector
  tasksMax: 1
  config:
    topics: my-topic
    value.converter: org.apache.kafka.connect.storage.StringConverter
    key.converter: org.apache.kafka.connect.json.JsonConverter
    key.converter.schemas.enable: false
    value.converter.schemas.enable: false
    camel.sink.endpoint.fileName: mydata-${date:now:ddMMyy-hhmmss-SSS}.txt
    camel.sink.path.directoryName: /opt/kafka/data
    camel.sink.endpoint.fileExist: Append
    camel.sink.endpoint.autoCreate: true
    camel.aggregation.disable: true
```

The KafkaConnector is configured to use the "org.apache.camel.kafkaconnector.file.CamelFileSinkConnector" class.

The value converter is set to "org.apache.kafka.connect.storage.StringConverter".

The connector filename is set using the expression: mydata-\${date:now:ddMMyy-hhmmss-SSS}.txt

When the connector starts your kafka records will be copied to the persistent volume as files:

```
docker@minikube:/var/strimzi/files$ ls -l
total 68
-rw-r--r-- 1 1001 root 142 Feb  1 09:58 mydata-010223-095832-818.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:58 mydata-010223-095832-996.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:58 mydata-010223-095833-103.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:58 mydata-010223-095833-111.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:58 mydata-010223-095837-438.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:58 mydata-010223-095857-469.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:59 mydata-010223-095917-485.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:59 mydata-010223-095937-511.txt
-rw-r--r-- 1 1001 root 142 Feb  1 09:59 mydata-010223-095957-538.txt
-rw-r--r-- 1 1001 root 142 Feb  1 10:00 mydata-010223-100017-585.txt
-rw-r--r-- 1 1001 root 141 Feb  1 10:00 mydata-010223-100037-606.txt
-rw-r--r-- 1 1001 root 142 Feb  1 10:06 mydata-010223-100638-700.txt
-rw-r--r-- 1 1001 root 142 Feb  1 10:06 mydata-010223-100642-123.txt
-rw-r--r-- 1 1001 root 142 Feb  1 10:07 mydata-010223-100702-150.txt
-rw-r--r-- 1 1001 root 142 Feb  1 10:07 mydata-010223-100722-158.txt
-rw-r--r-- 1 1001 root 142 Feb  1 10:07 mydata-010223-100742-179.txt
-rw-r--r-- 1 1001 root 142 Feb  1 10:08 mydata-010223-100802-208.txt
docker@minikube:/var/strimzi/files$ cat mydata-010223-100742-179.txt | xargs echo
{camelInfluxDB.MeasurementName:disk_space,time:2023-02-01T10:07:42Z,host:localhost,region:IE-region,used:19%,free:81%}
```

Links

[Kafka Connect And Schemas](#)

[Building your own Kafka Connect image with a custom resource](#)

[Kafka Connectors with Strimzi](#)

[Kafka Connect Deep Dive – Converters and Serialization Explained](#)

[Connector Developer Guide](#)

[How to Write a Connector for Kafka Connect – Deep Dive into Configuration Handling](#)

[Apache Camel](#)

[Camel Kafka Connector Examples](#)

[Camel Kafka Connect Maven Repository](#)

[Camel Basic Configuration](#)

[Camel Components Source Code](#)

[Camel Kafka Connector Source Code](#)

[Using secrets in Kafka Connect configuration](#)

[Kafka Connect Transformations](#)