Kafka Connect Reference Guide

15th June 2020

Table of Contents

1. Introduction

The Kafka integration for the Privitar Data Privacy Platform can be used to apply a Data Flow Job consuming the records from a Kafka topic and streaming the results to another Kafka topic. The Confluent "Gold Verified" integration is built on top of the Kafka Connect ecosystem. It comes as a Connector you will be able to install on an existing cluster or a dedicated cluster. You will be able to reuse the components you have already built for Kafka Connect such as the Converters and Transformations. It is tested to support the JsonConverter and AvroConverter provided by the Confluent Platform and can be integrated with the schema registry.

1.1. Compatibility

The plug-in is built to run on Kafka Connect 1.1 or later and is tested for the following combinations:

Kafka Connect version	Kafka version / Confluent version
Kafka Connect 1.1	Kafka 1.1 / Confluent 4.1
Kafka Connect 2.0	Kafka 1.1 / Confluent 5.0
Kafka Connect 2.4	Kafka 1.1 / Confluent 5.4

1.2. Deployment

The Kafka Integration for the Privitar Platform is a plug-in for Kafka Connect and can be deployed to an existing cluster. However, because of the nature of the work done by the connector, it can be more CPU intensive than other connectors. It is advised to deploy it to a cluster which will be dedicated to run the Privitar Data Flow jobs.

2. Architecture

The plugin is built with the following components:

- Consumer will pull the records from Kafka.
- Converter will deserialise binaries to a Kafka Connect data structure. This component is pluggable and could be any converter implementing the Kafka Connect api such as a StringConverter, AvroConverter or a JsonConverter.
- Transforms is a chain of transformations which enables to transform any fields in the Kafka Connect data structure before the data is processed by the Privitar Sink Connector. These transformations are pluggable and could be any transform implementing the Kafka Connect API. This can help to transform the types such as transforming a timestamp or a string to a Java Date or to flatten your data structure.
- Privitar Sink Connector will apply a Privitar Data Flow Job to the Record key and/or value. The key and value de-identification processes use different Data Flow Jobs.
- Destination Transforms is a chain of transformation similar to the first one, but can be used to transform back the Java types to a serialisable type after the data has been processed by the Privitar Sink Connector. This can help to transform the Java types such as Date to a formatted date string.
- Destination Converter will serialise the Kafka Connect data structure to binaries.
- Producer will push the records to Kafka.

3. Installation

This section describes the Installation procedure for the Privitar connector plug-in.

3.1. Pre-requisites

- Installed Kafka 1.1+ or Confluent 4.1+ (see supported versions below)
- Installed Kafka Connect 1.1+ (included from Confluent 4.1+).

3.2. Installation procedure

1. Create a folder in the plug-in directory used by the Kafka worker nodes.
2. Add the Privitar Connector plug-in to the new folder.
3. Add the necessary Token vault drivers into the new folder.
4. Repeat steps 2,3 and 4 on all Connect worker nodes
5. Restart all of the Connect worker nodes.

These steps need to be performed on every Kafka Connect worker node that will be using the Privitar plug-in.

3.3. Create a folder

Create a folder to store the Privitar connector plug-in. The location for plug-ins is defined by the `plugin.path` variable in the Kafka worker configuration file. For example:

```
plugin.path=/usr/local/share/kafka/plugins
```

Create a folder called, for example `privitar-kafka-connect` in that folder. This makes the path to the Privitar connector plug-in:

```
/usr/local/share/kafka/plugins/privitar-kafka-connect
```

3.4. Add the plug-in

Copy the Privitar connector plug-in JAR file into the new folder. The Privitar plug-in is supplied as an uber-JAR, so all the dependencies it requires are included in the jar - apart from the drivers that will be needed to connect to the Privitar Token vault.

3.5. Add Token Vault drivers

Drivers are required by the Privitar plug-in to connect to the Privitar Token vault. These drivers need to be added to the new Privitar plug-in folder. The drivers to include will be specific to the type of database you are using to store the Privitar Token vault.

3.6. Restart Worker nodes

Restart all Kafka Connect worker nodes from the console.

When the worker nodes are restarted they will discover all connectors, transforms, and/or converters defined within the Privitar plug-in. When we use a connector, transform, or converter, the Kafka Con-

nect worker loads the classes from the respective plug-in first, followed by the Kafka Connect runtime and Java libraries. Kafka Connect explicitly avoids all of the libraries in other plug-ins and prevents conflicts, making it very easy to use connectors and transforms developed independently by different providers.

4. Configuration using the REST API

To configure the Privitar plug-in using the REST API, follow the procedure below:

1. Make sure the Connector is properly installed on Kafka Connect by running:

```
curl -X GET http://localhost:8083/connector-plugins/
```

you should get back a list of connectors including the Privitar plug-in:

```
{"class":"com.privitar.agrotera.dataflow.kafka.PrivitarSinkConnector","type":"sink
","version":"3.2.0"}, ...]
```

2. Create a file `configuration.json` with your connector configuration:

```
{
"connector.class": "com.privitar.agrotera.dataflow.kafka.PrivitarSinkConnector",
"tasks.max": "2",
"topics": "my-topic",
"privitar.publisherUrl": "https://localhost:8080",
"privitar.publisherUsername": "admin",
"privitar.publisherPassword": "password",
"value.privitar.enabled": true,
"value.privitar.jobId": "8ebd",
"value.converter": "org.apache.kafka.connect.json.JsonConverter",
"value.converter.schemas.enable": false,
"dest.value.converter": "org.apache.kafka.connect.json.JsonConverter",
"dest.value.converter.schemas.enable": false,
"dest.bootstrap.servers": "localhost:9092",
"dest.topics": "my-destination-topic"
}
```

For more information on the configuration settings, see Configuration options [12].

3. Submit your new Connector to Kafka Connect:

```
curl -X GET http://localhost:8083/connectors/MyConnectorName/status
```

The response will show you if the Connector has actually started:

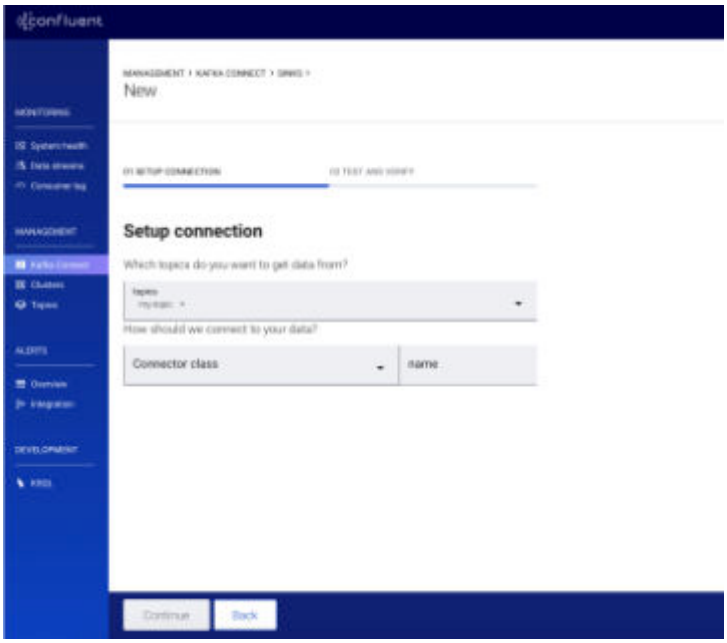
```
{
    "name": "MyConnectorName",
    "connector": {
        "state": "RUNNING",
        "worker_id": "connect:8083"
    },
    "tasks": [
        {
            "state": "RUNNING",
            "id": 0,
            "worker_id": "connect:8083"
        },
        {
            "state": "RUNNING",
            "id": 1,
            "worker_id": "connect:8083"
        }
    ],
    "type": "sink"
}
```

Finally, check the Kafka Connect logs where the plugin is running (the location depends on configuration, e.g. stdout).

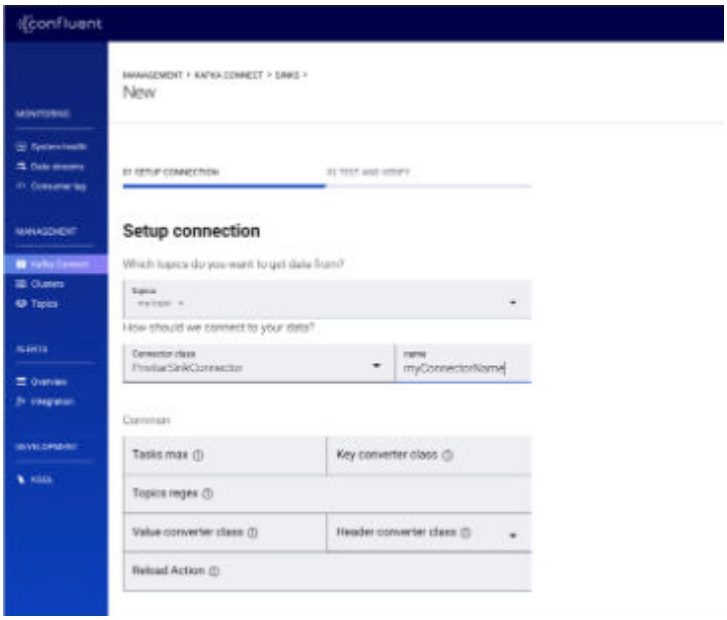
5. Configuration using Confluent Control Center

To configure the Privitar plug-in using the Confluent Control Center, follow the procedure below:

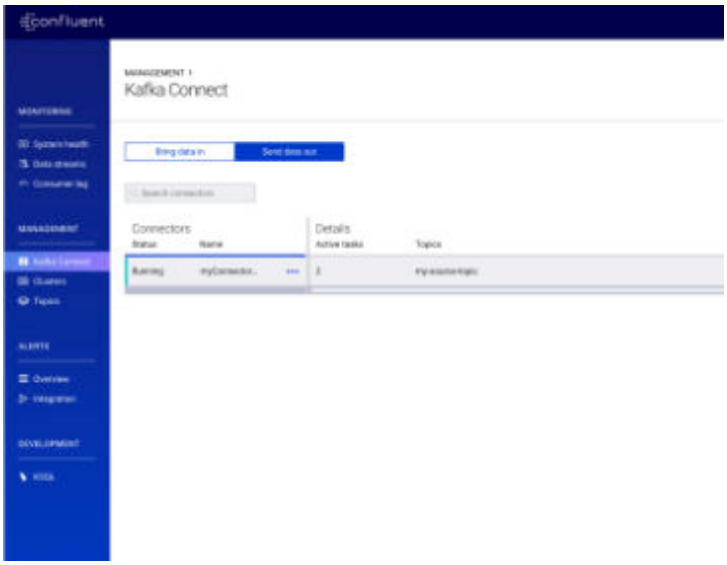
1. Select t **Kafka Connect** in the side bar.
2. Go to the **SINKS** tab.
3. Click **+ New sink**.
4. Select the topic from where you want to consume the data:



5. Click **Continue**.
6. Select the **Connector class** to be `PrivitarSinkConnector` and give a name to the Connector instance:



7. Add the configuration of the connector. (See Configuration Options [12] for more information on the configuration options.)
8. Click **Continue** to review the configuration before submitting the configuration to Kafka Connect.
9. You will be able to see the Connector running:



6. Configuration Options

The connector can be configured with the default SinkConnector settings and the following additional settings.

NOTE

In the Privitar Platform v3.1.1 and earlier, the properties starting with `privitar.*`, `value.privitar.*` and `key.privitar.*` used to start with `anonymiser.*`, `value.anonymiser.*` and `key.anonymiser.*`. From 3.2.0 it is possible to unmask values in a Data Flow Job by using the separate `UnmaskPrivitarSinkConnector` connector class.

The following sections describe the configuration options that are available.

6.1. Privitar platform

The following table defines the configuration options available for the Privitar platform.

Property Name	Description
key.privitar.enabled	Set to true if you want to enable the Data Flow Job on the record keys.
key.privitar.jobId	This is the Unique ID of the Data Flow Job to apply to the record keys.

key.privitar.schemaToJobIdMapping	This is a mapping of fully qualified Avro schema names to Data Flow Job IDs provided by the Privitar Policy Manager. The format should be:
	`<fully-qualified-schema-name>:<job-id>`
	in a comma-separated list. For example:
	`com.privitar.SchemaName1:3uhfkd,com.privitar.SchemaName2:4uj3ld`

value.privitar.enabled	Set to `True` if you want to enable the Data Flow Job on the record values.
value.privitar.jobId	This is the Unique ID of the Data Flow Job to apply to the record values.

value.privitar.schemaToJobIdMapping	This is a mapping of fully qualified Avro schema names to Data Flow Job IDs provided by the Privitar Policy Manager. The format should be:
	`<fully-qualified-schema-name>:<job-id>`
	For example:
	`com.privitar.SchemaName1:3uhfkd,com.privitar.SchemaName2:4uj3ld`

privitar.publisherUrl	The Privitar Platform host and port. For example: `http://localhost:8080`
privitar.publisherUsername	The Privitar Platform username.
privitar.publisherPassword	The Privitar Platform password.

6.2. Connector convertors

The following table defines the configuration options available for the Privitar Connector convertors.

Property Name	Description
dest.key.converter	The converter class to use to serialise the record key before to send them out.
dest.value.converter	The converter class to use to serialise the record value before to send them out.

6.3. Processing Guarantee

The following table defines the configuration options available for the Privitar Processing Guarantee.

Property Name	Description
processing.guarantee	The *processing guarantee* that should be used. Possible values are `exactly_once` (default) and `at_least_once`. Note that exactly-once processing requires a cluster of at least three brokers by default which is the recommended setting for production; for development you can change this, by adjusting broker setting `offset.state.log.replication.factor`.
transactional.id.prefix	This is the prefix the connector will use to generate the `transactional.id` in case `processing.guarantee=exactly_once`. Check Kafka documentation for more details about how to pick a `transactional.id`.

6.4. Error Handling

The following table defines the configuration options available for Error Handling.

Property Name	Description
dest.errors.handler	The failure handler to use. Possible values are: • `none` (default) • `dead_letter_queue` • `debug_log` Note that `debug_log` should only be chosen with non-confidential test data since it will expose the data that is sent. This handler will cover failures in Processor, Destination Transformations, Destination Converter and Producer. To cover failures on the first 2 stages 'Consumer', 'Converter' and 'Transformations', you will have to specify the Kafka connect built-in `errors.handler`.
dest.errors.dlq.topic.name	The topic name used if the error handler is `dead_letter_queue`. The failed records will be sent out to this topic. This is applicable for the following connector stages: Processor, Destination Transformations, Destination Converter and Producer. All errors in the stages before these ones won't be forwarded to this DLQ. However, there is a property called `errors.dlq.topic.name` in Kafka Connect you can enable to cover these other stages.
dest.errors.transforms	Aliases for the transformations to be applied to records sent to the DLQ. Similar to the one provided by Kafka Connect with ‘transforms’.
dest.errors.transforms.*	The configuration of the transformation applied before sending the failed input record to the DLQ. The configuration is similar to the 'transforms' setting from Kafka Connect.

6.5. Advanced Connector Settings

The following table defines the Advanced Connector Settings for the Privitar platform.

Property Name	Description
value.schema.singleFieldName	The field name in the Privitar schema to be used in case the record value is a simple String or Long (and not an object with multiple fields.
key.schema.singleFieldName	This is the prefix the connector will use to generate the `transactional.id` in case `processing.guarantee=exactly_once`. Check Kafka documentation for more details about how to pick a `transactional.id`.

Property Name	Description
dest.key.schema.name	The fully qualified name of the schema used for record keys "(eg. with Avro, it will be the namespace and the name of a record such as `com.record.namespace" .RecordName`). Default is the same schema name as the input record. This property is only valid in conjunction with the key.privitar.jobId property.
dest.value.schema.name	The fully qualified name of the schema used for record values "(eg. with Avro, it will be the namespace and the name of a record such as `com.record.namespace" .RecordName`). Default is the same schema name as the input record. This property is only valid in conjunction with the key.privitar.jobId property.
dest.key.schema.name.mapping	The mapping of the fully qualified name of the input schemas to the desired fully qualified name of the output schemas used for record keys. Default is the same schema name as the input record. This property is only valid in conjunction with the value.privitar.schemaToJobIdMapping property. The format should be: <fully-qualified-input-schema-name>:<fully-qualified-output-schema-name> in a comma-separated list. For example: com.privitar.SchemaName1:com.privitar.deidentified.SchemaName1, com.privitar.SchemaName2:com.privitar.deidentified.SchemaName
dest.value.schema.name.mapping	The mapping of the fully qualified name of the input schemas to the desired fully qualified name of the output schemas used for record values. Default is the same schema name as the input record. This property is only valid in conjunction with the key.privitar.schemaToJobIdMapping property. The format should be: <fully-qualified-input-schema-name>:<fully-qualified-output-schema-name> in a comma-separated list. For example: com.privitar.SchemaName1:com.privitar.deidentified.SchemaName1, com.privitar.SchemaName2:com.privitar.deidentified.SchemaName
dest.transforms	Aliases for the transformations to be applied to records. Similar to the one provided by Kafka Connect with ‘transforms’.
dest.transforms.*	The configuration of the transformation applied before to send the anonymised record. The configuration is similar to the 'transforms' setting from Kafka Connect.

6.6. Advanced Privitar Platform Settings

The following table defines advanced settings for the Privitar platform.

Property Name	Description
privitar.maxCacheWeightBytes	The maximum size (in bytes) that can be used by cached tokens.
privitar.maxBatchSize	Incoming records will be processed in batches no larger than this size.
privitar.numConcurrentBatches	The maximum number of batches that can be processed in parallel.
privitar.tokenVault.kerberosKeytabPath	Specifies the location of the kerberos keytab used for connecting to a HBase token vault.

7. Configuration Examples

Firstly, you will have to specify the common required attributes including the input topic from where will be consumed the messages:

```
connector.class=com.privitar.agrotera.dataflow.kafka.PrivitarSinkConnector
tasks.max=2
topics=my-topic
```

Then, you will have to specify the destination topic where the anonymised records will be produced. The Kafka Broker can be the same as where is sitting the input topic, but can also be different:

```
dest.bootstrap.servers=kafka-broker:9092
dest.topics=my-destination-topic
```

The required parameters to connect to the Privitar Policy Manager using Basic Authentication:

```
privitar.publisherUrl=https://privitar-policy-manager:8080
privitar.publisherUsername=myUsername
privitar.publisherPassword=mySecretPassword
```

Then, if you are using Confluent Schema Registry and Avro, you will have to setup the Kafka Connect converter for the record value and the Privitar Data Flow Job ID associated with record schema:

```
value.privitar.enabled=true
value.privitar.jobId=c21a
value.converter=io.confluent.connect.avro.AvroConverter
value.converter.schema.registry.url=http://schema-registry:8083
dest.value.converter=io.confluent.connect.avro.AvroConverter
dest.value.converter.schema.registry.url=http://schema-registry:808
```

Similarly, if you are using Confluent Schema Registry and Avro, but plan to process multiple different Avro Schema types for the given topic, you will have to set up the Kafka Connect converter for the record value and the Avro Schema Name to Job ID Mapping for each Schema type. You will also need to configure the a subject name strategy on the destination converter which supports different Schema types for the same topic (read more about the Schema Registry's subject name strategy).

```
value.privitar.enabled=true
value.privitar.schemaToJobIdMapping=com.privitar.SchemaName1:3uhfkd,com.privitar.Schem
aName2:4uj3ld
value.converter=io.confluent.connect.avro.AvroConverter
value.converter.schema.registry.url=http://schema-registry:8083
dest.value.converter=io.confluent.connect.avro.AvroConverter
dest.value.converter.schema.registry.url=http://schema-registry:8083
dest.value.converter.value.subject.name.strategy=io.confluent.kafka.serializers.subjec
t.TopicRecordNameStrategy
```

Alternatively, if you are using JSON without schema registry, you can setup a Kafka Connect converter with JSON as below:

```
value.privitar.enabled=true
value.privitar.jobId=c21a
value.converter=org.apache.kafka.connect.json.JsonConverter
value.converter.schemas.enable=false
dest.value.converter=org.apache.kafka.connect.json.JsonConverter
dest.value.converter.schemas.enable=false
```

If your messages have a record key, you can for example setup Kafka Connect converter for String and associate it with a different Privitar Data Flow Job ID. Note, if you are using StringConverter or Long-

Converter, you will have to create a Privitar Table with a single field (e.g. field1) and set the "singleFieldName" property as below.

```
key.privitar.enabled=true
key.privitar.jobId=4e5c key.schema.singleFieldName=field1
key.converter=org.apache.kafka.connect.storage.StringConverter
dest.key.converter=org.apache.kafka.connect.storage.StringConverter
```

In development environment, you might like to log failure records in Kafka Connect logs (you will need your Connect process to be logging Privitar code at least at INFO level to see failures: log4j.category.com.privitar=INFO):

```
dest.errors.handler=debug_log
```

In production environment, you should prefer to use a dead letter queue where to send failure records. The first 2 settings are to enable Kafka Connect built-in DLQ and the last 2 to enable the handler specific to the Privitar plugin. You should additionally specify the transforms to be applied; typically they would be the same as the input transforms.* settings, but with the *.target.type setting targeting the original input (eg JSON string) rather than the Java type (eg. Date):

```
errors.dlq.enable=true
errors.dlq.topic.name=my-dead-letter-queue
dest.errors.handler=dead_letter_queue
dest.errors.dlq.topic.name=my-dead-letter-queue
```

To unmask values, all the configuration options remain the same, but the connector class property should be set to:

```
connector.class=com.privitar.agrotera.dataflow.kafka.UnmaskPrivitarSinkConnector
```

8. Supported Data Types

This section describes the Converters used by the Privitar Connector plug-in together with the supported data types used by the Connector.

8.1. Converters

Converters are used to serialise and deserialise the data from a transportable data format to a Kafka Connect data structure.

This plugin is tested to work with the following converters:

Convertor	Path
Avro (Confluent platform)	`io.confluent.connect.avro.AvroConverter`
Json	`org.apache.kafka.connect.json.JsonConverter`
String	`org.apache.kafka.connect.storage.StringConverter`
Long	`org.apache.kafka.connect.storage.LongConverter`

However, it is possible to plug any converter as long it is implementing the Kafka Connect "Converter" interface.

As described in Configuration using the REST API [8], you will have to specify some converters to:

- Deserialise the key of consumed records (`key.converter`).
- Deserialise the value of consumed records (`value.converter`).
- Serialise the key of the anonymised records we want to send out (`dest.key.converter`).
- Serialise the value of the anonymised records we want to send out (`dest.value.converter`).

8.2. Data Types

The Privitar Data Flow supports the following data types:

Privitar Platform type	Java type accepted in input	Java type written in output
Boolean	Boolean	Boolean
Float	Float	Float
Double	Double	Double
Byte	Byte	Byte
Short	Short	Short
Integer	Integer	Integer
Long	Long	Long
Text	String	String
Date	java.util.Date	java.util.Date
Timestamp	java.util.Date	java.util.Date
Other	Any Java type	The same Java type that was input. This type is used when data is loaded from a source format that contains types that Privitar does not support. This enables data to be passed through unchanged or dropped by specifying Retain or Drop rules on the data in the Privitar policy. Downstream systems consuming the Privitar output can still use the data if it is retained by the Privitar policy.

Depending of which converter you use, you might have to add few transformations to fit your Kafka record data model with the Privitar schema.

For example, if you have a date/timestamp formatted as a string, you will have to add a transformation to transform it to a Java Date with properties such as (for timestamps, the type field will be Timestamp):

```
transforms=DateOfBirth
transforms.DateOfBirth.type=org.apache.kafka.connect.transforms.TimestampConverter
$Value
transforms.DateOfBirth.field=dateOfBirth
transforms.DateOfBirth.target.type=Date
transforms.DateOfBirth.format=yyyy-MM-dd
```

And transform this Java Date back to a formatted date string once the Data Flow Job has been applied:

```
dest.transforms=DateOfBirth
dest.transforms.DateOfBirth.type=org.apache.kafka.connect.transforms.TimestampConverte
r$Value
dest.transforms.DateOfBirth.field=dateOfBirth
dest.transforms.DateOfBirth.target.type=string
dest.transforms.DateOfBirth.format=yyyy-MM-dd
```

NOTE

org.apache.kafka.connect.transforms.TimestampConverter does not support NULL values.

9. Connecting with Kerberos

To use the Privitar Connector with Kerberos, there are three areas that you need to check and if necessary modify:

- Default Realm definition
- Connector settings
- Producer settings

WARNING
It is important to check these settings. If the Kerberos connection is not setup correctly, it will retry indefinitely, without issuing any error or warning message.

9.1. Default Realm

On each connector host, your default realm should be defined in the `/etc/krb5.conf` file.

9.2. Connector Settings

The Connector settings are defined in the `connect-distributed.properties` file. This file is used to start the Connect instances and would be expected to have the following properties and configured to use one of the following three security protocols:

- SASL_PLAINTEXT
- SASL_SSL

WARNING
It is important to check these settings. If the Kerberos connection is not setup correctly, it will retry indefinitely, without issuing any error or warning message.

9.2.1. Connector settings (using SASL_PLAINTEXT security protocol)

The `connect-distributed.properties` file that is used to start the connect instances would be expected to have the following properties set (since other connectors would need to leverage the Kerberos connection).

```
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
# Configure SASL_SSL if SSL encryption is enabled, otherwise configure SASL_PLAINTEXT
security.protocol=SASL_PLAINTEXT
```

```
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
useKeyTab=true \
keyTab="/path/to/keytab/file/kafka.keytab"\
storeKey=true \
useTicketCache=false \
serviceName="kafka" \
principal="kafka/a-full-host-name@APRINCIPALNAME.COM";

producer.sasl.mechanism=GSSAPI
producer.sasl.kerberos.service.name=kafka
# Configure SASL_SSL if SSL encryption is enabled, otherwise configure SASL_PLAINTEXT
producer.security.protocol=SASL_PLAINTEXT
producer.sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
useKeyTab=true \
keyTab="/path/to/keytab/file/kafka.keytab" \
storeKey=true \
useTicketCache=false \
serviceName="kafka" \
principal="kafka/a-full-host-name@APRINCIPALNAME.COM";

consumer.sasl.mechanism=GSSAPI
consumer.sasl.kerberos.service.name=kafka
# Configure SASL_SSL if SSL encryption is enabled, otherwise configure SASL_PLAINTEXT
consumer.security.protocol=SASL_PLAINTEXT
consumer.sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
useKeyTab=true \
keyTab="/path/to/keytab/file/kafka.keytab" \
storeKey=true \
useTicketCache=false \
serviceName="kafka" \
principal="kafka/a-full-host-name@APRINCIPALNAME.COM";
```

9.2.2. Connector settings (using SASL_SSL security protocol)

To use SASL_SSL as the security protocol between clients/connectors and the brokers instead of SASL_PLAINTEXT, use the following configuration in addition to what is above (some properties need to be replaced):

```
security.protocol=SASL_SSL
consumer.security.protocol=SASL_SSL
producer.security.protocol=SASL_SSL

ssl.keystore.location=/path/to/a/client/keystore/file/kafka.client.keystore.jks
ssl.keystore.password=client-keystore-password
ssl.truststore.location=/path/to/a/client/truststore/file/kafka.client.truststore.jks
ssl.truststore.password=client-truststore-password
ssl.key.password=client-keystore-password

consumer.keystore.location=/path/to/a/client/keystore/file/kafka.client.keystore.jks
consumer.keystore.password=client-keystore-password
consumer.truststore.location=/path/to/a/client/truststore/file/
kafka.client.truststore.jks
consumer.truststore.password=client-truststore-password
consumer.key.password=client-keystore-password

producer.keystore.location=/path/to/a/client/keystore/file/kafka.client.keystore.jks
producer.keystore.password=client-keystore-password
producer.truststore.location=/path/to/a/client/truststore/file/
kafka.client.truststore.jks
```

```
producer.truststore.password=client-truststore-password
producer.key.password=client-keystore-password
```

9.3. Producer settings

The Privitar Connector uses its own Producer, so Kerberos only needs to be configured for the Producer in the task config. Since this is a Sink Connector, the Consumer side of things is handled by the Connect cluster default Kerberos properties described in the previous section. This section provides examples of Producer settings using the following security protocols:

- SASL_PLAINTEXT - auth but no encryption
- SASL_SSL - auth with kerberos and encrypted comms

WARNING

It is important to check these settings. If the Kerberos connection is not setup correctly, it will retry indefinitely, without issuing any error or warning message.

9.3.1. Producer settings (using SASL_PLAINTEXT security protocol)

The producer configuration should look as follows:

NOTE

For the dest.sasl.jaas.config property, make sure there are no spaces between lines of the jaas.config (i.e. where there are \n, there should be no spaces before or after)

```
"dest.bootstrap.servers": "1.1.1.1:9092,1.1.1.2:9092,1.1.1.3:9092",
"dest.sasl.mechanism":"GSSAPI",
"dest.sasl.kerberos.service.name":"kafka",
"dest.security.protocol":"SASL_PLAINTEXT",
"dest.sasl.jaas.config":"com.sun.security.auth.module.Krb5LoginModule required
\nuseKeyTab=true\nkeyTab=\"/path/to/keytab/file/kafka.keytab\"\nstoreKey=true
\nuseTicketCache=false\nserviceName=\"kafka\"\nprincipal=\"kafka/a-full-host-
name@APRINCIPALNAME.COM\";"
```

9.3.2. Producer settings (using SASL_SSL security protocol)

To use SASL_SSL as the security protocol between the connector and the brokers instead, use the following configuration in addition to what is above (some properties need to be replaced):

```
"dest.security.protocol":"SASL_SSL",
"dest.ssl.keystore.location":"/path/to/a/client/keystore/file/
kafka.client.keystore.jks",
"dest.ssl.keystore.password":"client-keystore-password",
"dest.ssl.truststore.location":"/path/to/a/client/truststore/file/
```

```
kafka.client.truststore.jks",
"dest.ssl.truststore.password":"client-truststore-password",
"dest.ssl.key.password":"client-keystore-password"
```

NOTE

For the dest.bootstrap.servers property, make sure to use the right port for the SASL_SSL protocol.