



Kafka Connect

Reference Guide

Publication date : June 4, 2021

All rights reserved.

Table of Contents

1. Introduction	4
1.1. Compatibility	4
1.2. Deployment	4
2. Architecture	5
3. Installation	6
3.1. Pre-requisites	6
3.2. Installation procedure	6
3.3. Create a folder	6
3.4. Add the plug-in	6
3.5. Add Token Vault drivers	6
3.6. Restart Worker nodes	7
4. Configuring users	8
4.1. Creating an API user to run Data Flow jobs	8
4.2. Masking Jobs	9
4.3. UnMasking Jobs	10
5. Configuration using the REST API	12
6. Configuration using Confluent Control Center	14
7. Configuration Options	17
7.1. Privitar platform	17
7.2. Connector converters	18
7.3. Processing Guarantee	18
7.4. Error Handling	18
7.5. Advanced Connector Settings	19
7.6. Advanced Privitar Platform Settings	20
8. Configuration Examples	21
9. Supported Data Types	23
9.1. Converters	23
9.2. Data Types	23
10. Connecting with Kerberos	25
10.1. Default Realm	25
10.2. Connector Settings	25
10.2.1. Security and Authentication	25
10.2.2. Configuration	25
10.2.3. Producer settings	26

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.

By default, Kafka makes a connection to Privitar using Basic authentication, but Privitar also supports Mutual TLS authentication.

The integration 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 Privitar connector is built to run on Kafka Connect 2.0 or later and is tested for the following combinations with the Privitar platform v3.8 (or later):

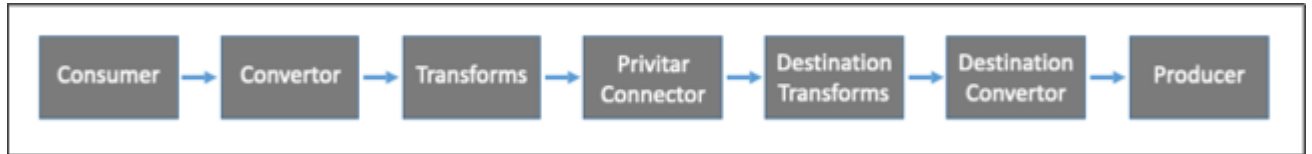
Kafka Connect version	Kafka version / Confluent version
Kafka Connect 2.0	Kafka 2.0 / Confluent 5.0
Kafka Connect 2.0.x	Kafka 2.3.x / Confluent 5.3

1.2. Deployment

The Privitar connector can be deployed to an existing cluster. However, because of the nature of the work performed 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 Privitar Data Flow jobs.

2. Architecture

The Privitar Connector is built with the following components:



- Consumer will pull the records from Kafka.
- Converter will deserialize 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 can 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 the data structure.
- The Privitar Connector will apply a Privitar Data Flow Job to the Record key and/or value. The Data Flow Job can be a Masking Data Flow Job or an UnMasking Data Flow Job.
- Destination Transforms is a chain of transformation similar to the first one, but can be used to transform back the Java types to a serializable type after the data has been processed by the Privitar Connector. This can help to transform the Java types such as Date to a formatted date string.
- Destination Converter will serialize 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.

The Connector consists of two separate plug-ins, but they are both contained in single jar:

- Masking plug-in connector. (This plug-in is called, `PrivitarSinkConnector`)
- Unmasking plug-in connector. (This plug-in is called, `UnmaskPrivitarSinkConnector`)

This document will refer to the *Privitar Connector plug-in* to mean both plug-ins. Where necessary, it will reference a specific plug-in.

3.1. Pre-requisites

- Installed Kafka 1.1+ or Confluent 4.1+ (see supported versions in [Introduction \[4\]](#))
- 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 node.
2. Add the Privitar Connector plug-in to the new folder.
3. Add the necessary Token vault drivers into the new folder.
4. Restart the Connect worker node.

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 Connect 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. Configuring users

To run Data Flow jobs in Privitar you need to create API users with the correct permissions to run both Masking and UnMasking Jobs in the Team that the Data Flow Job is defined in. This section describes how to create and configure API users on Privitar to run Data Flow Jobs. It is applicable for Data Flow jobs being set up on any of the following data processing platforms:

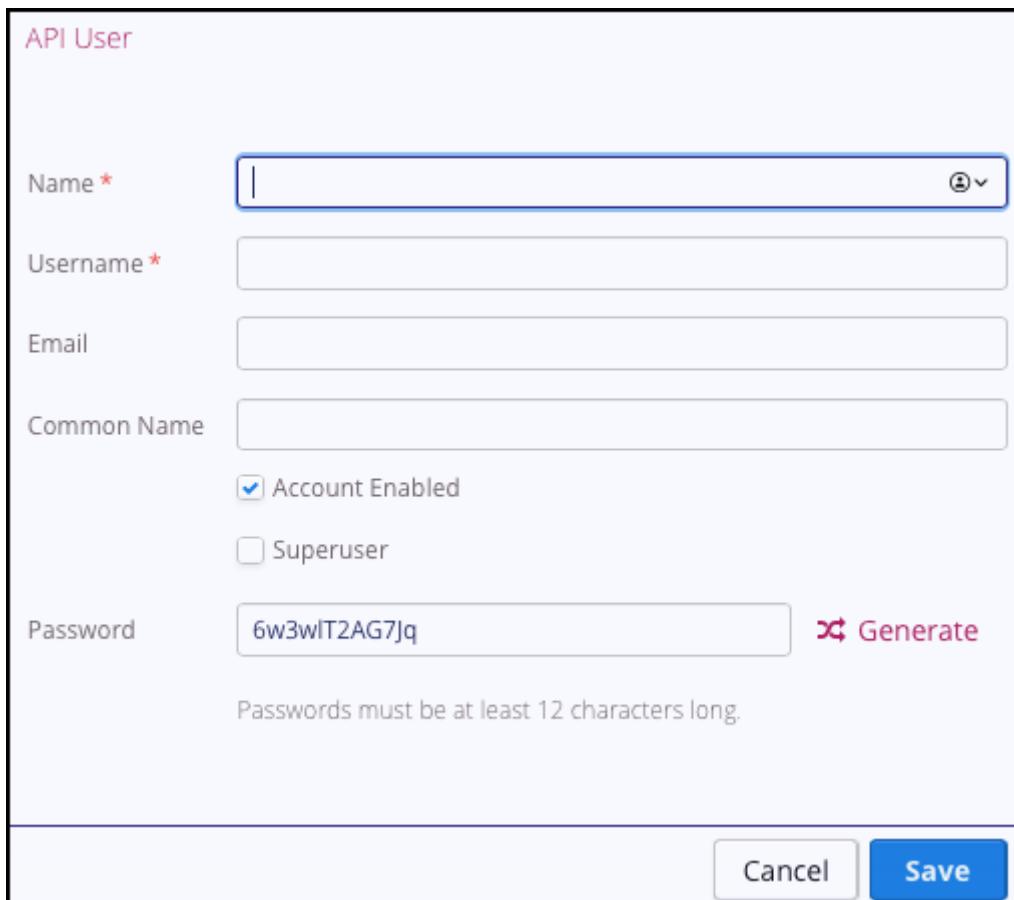
- Kafka/Confluent
- Apache Nifi
- StreamSets

For more general information about managing Users, Roles and Teams in Privitar, refer to the *Privitar Data Privacy Platform User Guide*.

4.1. Creating an API user to run Data Flow jobs

To create API users in Privitar for running Data Flow jobs:

1. Select **API Users** from the Superuser navigation panel.
2. Select **Create New API User**. The **API User** dialog box is displayed:

The image shows a dialog box titled "API User" with a light blue background. It contains several input fields: "Name" (with a red asterisk and a dropdown icon), "Username" (with a red asterisk), "Email", "Common Name", and "Password" (with a "Generate" button). There are also two checkboxes: "Account Enabled" (checked) and "Superuser" (unchecked). At the bottom, there are "Cancel" and "Save" buttons. A note at the bottom states "Passwords must be at least 12 characters long." The "Name" field is currently empty, while the "Username" field is empty, "Email" is empty, "Common Name" is empty, and "Password" contains the text "6w3wIT2AG7Jq".

API User

Name *

Username *

Email

Common Name

☒ Account Enabled

☐ Superuser

Password [Generate](#)

Passwords must be at least 12 characters long.

Enter the details for the new API user. The first two fields - **Name** and **Username** - are mandatory. All other fields are optional:

- **Name** is the display name of the API user.
- **Username** is the unique username for the API user.
- **Email** is the email address associated with the API user. This is an optional field.
- **Common Name** is used for API authentication (if your Privitar installation is configured to use Mutual TLS) or **Password** (if basic HTTP authentication is used).
You can click on **Generate** to generate a new password.
- To make sure the User account is activated, select the **Account Enabled** check box.
- Optionally, if you want this new API User to have Superuser permissions, select the **Superuser** check box.

3. Click **Save** to save the details entered and to create the API user. The new API user will be added to the list of API users shown in the main window.

Typically, you would create two API users; one to run Masking jobs and the other to run UnMasking jobs. It is also possible for a single API user to run both jobs if required.



NOTE

It is also possible in Privitar to manage users externally in LDAP. If managing users in this way, then instead of assigning individual API users to a team role, you need to assign an LDAP group to the relevant team role.

4.2. Masking Jobs

By default, the **Data Flow Operator** Role in the **default** Team in Privitar has the **Run Data Flow** permission enabled for **Masking Jobs**. See:

Edit Role

Name *

Read Permissions All of a team's objects are automatically visible to users who have any role in that team

Write Permissions

Object	Actions
Schemas	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
Policies	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
Rules	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
Masking Jobs	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Cancel <input type="checkbox"/> Run Batch <input checked="" type="checkbox"/> Run Data Flow <input type="checkbox"/> Run POD
Unmasking Jobs	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Cancel <input type="checkbox"/> Run Batch <input type="checkbox"/> Run Data Flow <input type="checkbox"/> Run POD
Protected Data	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Close <input type="checkbox"/> Unmask Token <input type="checkbox"/> Run Unveiler <input type="checkbox"/> Run Remasking
Environments	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Test <input type="checkbox"/> Match Watermark

Cancel Save

The API user that has been created in Privitar will need to be assigned the role of **Data Flow Operator** in the Team that the job is defined in.

In the example below, an API user called **data_ops_api_user** has been created and assigned the role of **Data Flow Operator**:

Team default

Roles

All available roles

Role	Users
Admin	0
All permissions	20
Author	0
Create PDD only	0
Create new role while p	0
Data Flow Operator	1
Environments Editor	0
Investigator	0
Operator	0
Policy Delete	0
Run batch job only	0
Schema Creator Only	0

Users

Users belonging to the selected role

Use the slider to reveal the full name

Name	Username	Email	Actions
Data Ops	data_ops_api_user		Remove

Add User

Cancel Save

4.3. UnMasking Jobs

For UnMasking jobs, you need to assign an API user to a Role that has permission to Run Data Flow UnMasking jobs in the Team that the job is defined in.

In the example below, a new Role has been created called, **Data Flow (Unmasking)** with the **Run Data Flow** permission enabled for **Unmasking jobs**:

Edit Role

Name *

Read Permissions All of a team's objects are automatically visible to users who have any role in that team

Write Permissions

Object	Actions
Schemas	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
Policies	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
Rules	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
Masking Jobs	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Cancel <input type="checkbox"/> Run Batch <input type="checkbox"/> Run Data Flow <input type="checkbox"/> Run POD
Unmasking Jobs	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Cancel <input type="checkbox"/> Run Batch <input checked="" type="checkbox"/> Run Data Flow <input type="checkbox"/> Run POD
Protected Data	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Close <input type="checkbox"/> Unmask Token <input type="checkbox"/> Run Unveiler <input type="checkbox"/> Run Remasking
Environments	<input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete <input type="checkbox"/> Test <input type="checkbox"/> Match Watermark

Cancel Save

The new additional API user (**data_ops2_api_user**) can be assigned to the **Data Flow (Unmasking)** role:

Team default

Roles

All available roles


Role	Users
Admin	0
All permissions	20
Author	0
Create PDD only	0
Create new role while p	0
Data Flow (UnMasking)	1
Data Flow Operator	1
Environments Editor	0
Investigator	0
Operator	0
Policy Delete	0
Run batch job only	0

Users

Users belonging to the selected role

 Add User

Use the slider to reveal the full name

Name	Username	Email	Actions
Data Ops2	data_ops2_api_user		 Remove

Cancel

Save

5. 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 both Privitar plug-ins:

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

2. Create a file configuration.json file with your connector configuration for each plug-in. The example below shows the configuration for the PrivitarSinkConnector plug-in:

```
{
  "connector.class":
    "com.privitar.agrotera.dataflow.kafka.PrivitarSinkConnector",
  "tasks.max": "2",
  "topics": "my-topic",
  "privitar.publisherUrl": "https://localhost:8080",
  "privitar.publisherUsername": "api_user",
  "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 \[17\]](#).

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 plug-in is running. (The location depends on configuration. For example, `stdout`).

6. Configuration using Confluent Control Center

To configure the Privitar plug-in using the Confluent Control Center, follow the procedure below. This procedure must be followed for both Privitar Connector plug-ins:

- `PrivitarSinkConnector`
- `UnmaskPrivitarSinkConnector`

The procedure below is for the `PrivitarSinkConnector` plug-in. The same procedure needs to be repeated for the `UnmaskPrivitarSinkConnector` plug-in:

1. Select **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:

The screenshot shows the Confluent Control Center interface for creating a new sink. The left sidebar contains navigation links for MONITORING, MANAGEMENT, ALERTS, and DEVELOPMENT. The main content area is titled 'New' and shows a progress bar with two steps: '01 SETUP CONNECTION' (active) and '02 TEST AND VERIFY'. The 'Setup connection' section asks 'Which topics do you want to get data from?' and provides a dropdown menu with 'topics' and 'my-topic'. Below this, it asks 'How should we connect to your data?' and provides a table with two columns: 'Connector class' and 'name'. At the bottom of the form are 'Continue' and 'Back' buttons.

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

MANAGEMENT > KAFKA CONNECT > SINKS > New

01 SETUP CONNECTION 02 TEST AND VERIFY

Setup connection

Which topics do you want to get data from?

topics
my-topic

How should we connect to your data?

Connector class
PrivitarSinkConnector

name
myConnectorName

Common

Tasks max ⓘ	Key converter class ⓘ
Topics regex ⓘ	
Value converter class ⓘ	Header converter class ⓘ
Reload Action ⓘ	

7. Add the configuration of the connector. (See [Configuration Options \[17\]](#) 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:

The screenshot shows the Confluent Kafka Connect management interface. The left sidebar contains navigation links for MONITORING (System health, Data streams, Consumer lag), MANAGEMENT (Kafka Connect, Clusters, Topics), ALERTS (Overview, Integration), and DEVELOPMENT (KSQL). The main content area is titled 'Kafka Connect' and includes buttons for 'Bring data in' and 'Send data out', a search bar, and a table of connectors.

Connectors		Details	
Status	Name	Active tasks	Topics
Running	myConnector...	2	my-source-topic

7. 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.

7.1. Privitar platform

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

Property Name	Description
<code>key.privitar.enabled</code>	Set to <code>True</code> if you want to enable the Data Flow Job on t
<code>key.privitar.jobId</code>	This is the Unique ID of the Data Flow Job to apply to th
<code>key.privitar.schemaToJobIdMapping</code>	<p>This is a mapping of fully qualified Avro schema names IDs provided by the Privitar Policy Manager. The format</p> <pre><fully-qualified-schema-name>:<job-id></pre> <p>in a comma-separated list. For example:</p> <pre>com.privitar.SchemaName1:3uhfkd,com.privitar.Sch</pre>
<code>value.privitar.enabled</code>	Set to <code>True</code> if you want to enable the Data Flow Job on t
<code>value.privitar.jobId</code>	This is the Unique ID of the Data Flow Job to apply to th
<code>value.privitar.schemaToJobIdMapping</code>	<p>This is a mapping of fully qualified Avro schema names IDs provided by the Privitar Policy Manager. The format</p> <pre><fully-qualified-schema-name>:<job-id></pre> <p>For example:</p> <pre>com.privitar.SchemaName1:3uhfkd,com.privitar.Sch</pre>
<code>privitar.publisherUrl</code>	<p>The Privitar Platform host and port. For example:</p> <pre>http://localhost:8080</pre>
<code>privitar.publisherUsername</code>	<p>The username and password of the API user.</p> <p>The API user must have a role with Run Data Flow perm Masking jobs or Unmasking jobs in the team that the jo</p>

Property Name	Description
privitar.publisherPassword	For more information about configuring users, see Connections .
privitar.authentication	The method used to authenticate with the Privitar Policy Manager. Possible values are <code>mutualTls</code> and <code>basic</code> . The default setting is <code>basic</code> authentication.
privitar.tlsClientCertificatePath	The location of the certificate file used for authenticating to the Privitar Policy Manager.
privitar.tlsClientCertificatePassword	The password for the TLS client certificate file.
privitar.tlsTrustedCertificateAuthorityCertificatePath	The location of the TLS CA certificate file used for authenticating to the Privitar Policy Manager.
privitar.tlsHostnameVerification	Set to <code>True</code> (by default) to enable hostname verification for all connections to Privitar Policy Manager. This property should be enabled in most cases. Disabling verification will degrade the overall security of TLS as there is no guarantee about the server identity.

7.2. Connector converters

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

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.

7.3. Processing Guarantee

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

Property Name	Description
processing.guarantee	The <i>processing guarantee</i> that should be used. Possible values are <code>exactly_once</code> (default) and <code>at_least_once</code> . Note that <code>exactly_once</code> 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 <code>offset.state.log.replication.factor</code> .
transactional.id.prefix	This is the prefix the connector will use to generate the <code>transactional.id</code> in case <code>processing.guarantee=exactly_once</code> . Check Kafka documentation for more details about how to pick a <code>transactional.id</code> .

7.4. Error Handling

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

Property Name	Description
dest.errors.handler	<p>The failure handler to use. Possible values are:</p> <ul style="list-style-type: none"> • none (default) • dead_letter_queue • debug_log <p>Note that <code>debug_log</code> 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 <code>errors.handler</code>.</p>
dest.errors.dlq.topic.name	<p>The topic name used if the error handler is <code>dead_letter_queue</code>. 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 <code>errors.dlq.topic.name</code> in Kafka Connect you can enable to cover these other stages.</p>
dest.errors.transforms	<p>Aliases for the transformations to be applied to records sent to the DLQ. Similar to the one provided by Kafka Connect with 'transforms'.</p>
dest.errors.transforms.*	<p>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.</p>

7.5. 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 <code>transactional.id</code> in case <code>processing.guarantee=exactly_once</code> . Check Kafka documentation for more details about how to pick a <code>transactional.id</code> .
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 <code>`com.record.namespace`.RecordName`</code>). Default is the same schema name as the input record. This property is only valid in conjunction with the <code>key.privitar.jobld</code> 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 <code>`com.record.namespace`.RecordName`</code>). Default is the same schema name as the input record. This property is only valid in conjunction with the <code>key.privitar.jobld</code> property.

Property Name	Description
dest.key.schema.name.mapping	<p>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:</p> <p><fully-qualified-input-schema-name>:<fully-qualified-output-schema-name></p> <p>in a comma-separated list. For example:</p> <p>com.privitar.SchemaName1:com.privitar.deidentified.SchemaName1, com.privitar.SchemaName2:com.privitar.deidentified.SchemaName</p>
dest.value.schema.name.mapping	<p>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:</p> <p><fully-qualified-input-schema-name>:<fully-qualified-output-schema-name></p> <p>in a comma-separated list. For example:</p> <p>com.privitar.SchemaName1:com.privitar.deidentified.SchemaName1, com.privitar.SchemaName2:com.privitar.deidentified.SchemaName</p>
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.

7.6. 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.

8. Configuration Examples

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

```
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 anonymized 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.authentication=basic
privitar.publisherUrl=https://privitar-policy-manager:8080
privitar.publisherUsername=<myUsername>
privitar.publisherPassword=<mySecretPassword>
```

Alternatively, you can add the required parameters to connect to the Privitar Policy Manager using Mutual TLS:

```
privitar.authentication=mutualTls
privitar.tlsClientCertificatePath=<myClientPath>
privitar.tlsClientCertificatePassword=<myClientPassword>
privitar.tlsTrustedCertificateAuthorityCertificatePath=<myTrustedPath>
```

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.SchemaName2:4uj3ld
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.serialize
rs.subject.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 LongConverter, 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 a 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 a 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
```

9. 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.

9.1. 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:

Converter	Path
Avro (Confluent platform)	<code>io.confluent.connect.avro.AvroConverter</code>
Json	<code>org.apache.kafka.connect.json.JsonConverter</code>
String	<code>org.apache.kafka.connect.storage.StringConverter</code>
Long	<code>org.apache.kafka.connect.storage.LongConverter</code>

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 \[12\]](#), 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`).

9.2. 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	<code>java.util.Date</code>	<code>java.util.Date</code>

Privitar Platform type	Java type accepted in input	Java type written in output
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.TimestampConverter$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.

For more information about how to use the Confluent Timestamp converter, see the **TimeStampConverter** section in the [Confluent Technical Documentation](#).

10. Connecting with Kerberos

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

- Default Realm definition
- Connector 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.

10.1. Default Realm

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

10.2. 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 security and/or authentication protocols configured.

10.2.1. Security and Authentication

The Privitar Connector should be configured to use one of the following three security and/or authentication protocols:

- SASL_PLAINTEXT
- SASL_SSL
- 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.

10.2.2. Configuration

The Privitar Connector acts as both a Sink Connector (Consumer) as well as a Source connector (Producer). Therefore, you need to add both definitions for this Connector.

- The Sink Connector (Consumer) can use the definitions used by all Connectors in the `connect.distributed.properties` file.
- The Source Connector (Producer) definitions need to be added to the `configuration.json` file for the connector. The configuration statements are added using the `dest` prefix. Some examples are provided in the following section.

For more information about setting up Kerberos together with any additional encryption or security authentication protocol, refer to the Confluent documentation. Here are some appropriate links:

- https://docs.confluent.io/current/tutorials/security_tutorial.html
- <https://docs.confluent.io/current/kafka/encryption.html>
- https://docs.confluent.io/current/kafka/authentication_sasl/index.html

10.2.3. Producer settings

A column can also be configured as Do not generalise. If you choose this option, you must also set this as a sensitive column by selecting the Sensitive check box. Specifying a column as Sensitive and Do not generalise ensures that for each cluster of rows with the same quasi-identifier values, there is a diverse mix of values for the sensitive columns. See Sensitive fields and L-diversity for more information.

- 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.

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\" ; "
```

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.