



# pubsubplus-connector-spark

## *User Guide*

Solace Corporation

Version 3.1.1

**solace.**

# Table of Contents

Preface .....	1
Getting Started .....	2
Prerequisites .....	2
Supported Platforms .....	2
Quick Start common steps .....	2
Databricks Considerations .....	4
Checkpointing & Acknowledgement .....	5
Checkpoint Handling .....	5
User Authentication .....	5
Using Databricks Secret Management .....	6
Message Replay .....	7
Solace Spark Streaming Source Schema Structure .....	7
Solace Spark Streaming Sink Schema Structure .....	8
Configuration .....	10
Solace Spark Connector Source Configuration Options .....	10
Solace Spark Connector Sink Configuration Options .....	17
License .....	22
Support .....	22

# Preface

Solace PubSub+ Connector for Spark streams data from Solace PubSub+ broker to Spark Data Sources.

# Getting Started

This guide assumes you are familiar with Spark set up and Spark Structured Streaming concepts. In the following sections we will show how to set up Solace Spark Connector to stream data from Solace to Spark and publish events from Spark to Solace.

## Prerequisites

- [Solace PubSub+ Event Broker](#)
- Apache Spark 3.5.1 and Scala 2.12

## Supported Platforms

The connector is built on the Spark Structured Streaming API and has been tested on Azure Databricks. Since the Databricks runtime is consistent across all supported cloud platforms(AWS & Google Cloud), it is expected to behave similarly in other Databricks environments. Additionally, the connector has been validated on vanilla Apache Spark, ensuring compatibility with any platform that supports standard Spark deployments.

## Quick Start common steps

This quick start shows how to deploy connector in Databricks Cluster.

Solace Spark Connector is available in maven central with below coordinates.

```
<dependency>
  <groupId>com.solacecoe.connectors</groupId>
  <artifactId>pubsubplus-connector-spark</artifactId>
  <version>3.1.1</version>
</dependency>
```

Steps to deploy the connector

1. Create a Databricks Cluster with Spark Version 3.5.1 and Scala 2.12
2. Once the cluster is started choose libraries and select maven as connector source. Enter above maven coordinates to download connector from maven central.



Before installing latest version of connector make sure earlier versions of solace spark connector are completely deleted from cluster. This is to ensure there are no version conflicts to start the connector.

1. Create new scala notebook and copy below code snippets.
2. Creating Solace Source Streaming Queries
  - a. First, let's create a Solace consumer to read messages and write to parquet file. Update

configuration options as per your environment.

```

    val spark = SparkSession.builder.appName("SolaceSparkStreaming").
getOrCreate()
    val streamName = "solace-spark-connector-sample-test"
    val df = spark.readStream.format("solace")
      .option("host", "tcp://localhost:55555")
      .option("vpn", "default")
      .option("username", "default")
      .option("password", "default")
      .option("queue", "<queue-name>")
      .option("connectRetries", 2)
      .option("reconnectRetries", 2)
      .option("batchSize", 100)
      .option("includeHeaders", true)
      .option("partitions", 1)
      .load()

    // write to parquet file
    val query = df.writeStream
      .format("parquet")
      .outputMode("append")
      .queryName(streamName)
      .option("checkpointLocation",
s"/spark_checkpoints/solace_spark_connector_checkpoint/$streamName/")
      .option("path", s"/solace_spark_connector_parquet_output/$streamName/")
      .start()

    query.awaitTermination()

```

b. Finally, let's read data from the parquet file from the location configured above

```

    val streamName = "solace-spark-connector-sample-test" // this should be
equal to stream name variable provided as above
    val df = spark.read.format("parquet").load(
s"/solace_spark_connector_parquet_output/$streamName/") // this should be same
as value of "path" property configured in write stream as above
    display(df)
    // Ex: Parse payload as string
    df.select($"payload".cast("STRING"))

```

### 3. Creating Solace Sink Streaming Queries

a. First, let's create a parquet consumer to read messages and publish to Solace. Update configuration options as per your environment.

```

    val parquetData = spark.read.parquet("<path-to-parquet-file>")
    val struct_stream = spark.readStream

```

```

.schema(parquetData.schema)
.parquet("<path-to-parquet-file>")

// write to parquet file
val query = df.writeStream
.foreachBatch((outputDf: Dataset[org.apache.spark.sql.Row], bid: Long) => {
  // Process valid data frames only
  if (!outputDf.isEmpty) {
    // business logic
    try {
      outputDf.show()
      outputDf.write.format("solace")
        .option("host", "tcp://localhost:5555")
        .option("vpn", "default")
        .option("username", "default")
        .option("password", "default")
        .option("batchSize", outputDf.count())
        .option("topic", "solace/spark/publish") // This can be
commented if topic column is present in output dataframe.
        .option("id", "<application-message-id>") // This can be
commented if Id column is present in output dataframe.
        .mode(SaveMode.Append)
        .save()
    } catch {
      case e: Exception => {
        // Any logic to handle failed messages. Connector will log
application message id along with exception as json object.
        throw new RuntimeException(e)
      }
    }
  }
}).start()

query.awaitTermination()

```

- b. Finally, create a queue on Solace Broker and subscribe to the topic to spool the messages published by Solace Spark Connector.



Above sample code used parquet as example data source. You can configure your required data source to write data.



In case of databricks deployment, it is recommended to store and retrieve sensitive credentials from Databricks secrets. Please refer to [Using Databricks Secret Management](#) on how to configure secrets and use them in notebook.

## Databricks Considerations

In case if you are using Shared compute cluster, make sure your cluster has [appropriate permissions](#) to install connector from maven central and access the jars. Please contact your

Databricks administrator for required permissions.

## Checkpointing & Acknowledgement

Solace Spark connector relies on Spark Checkpointing mechanism to resume from last successful message. Connector acknowledges & writes processed message id's to spark checkpoint whenever Spark sends commit signal to connector. Commit signal is an indication by Spark that data has been processed successfully.

## Checkpoint Handling

Starting from version 3.1.0 connector, solace connection is now executed on worker node instead of driver node. This give us the ability to utilize cluster resource efficiently and also improves processing performance. The connector uses Solace LVQ to communicate checkpoint information from worker nodes to driver node(commit to checkpoint location) as they run on different JVM's.



In case of recovery, connector uses offset state from LVQ to identify last successfully processed messages. Hence, it is recommended not to delete or modify offset state in LVQ.

In some cases, there might be checkpoint failures as spark may fail to write to checkpoint during instance crash or unavailability or other reasons. Though the connector will handle duplicates in most cases, we recommend to keep your downstream systems idempotent.

## User Authentication

Solace Spark Connector supports Basic, Client Certificate and OAuth authentication to Solace. Client Credentials flow is supported when connecting using OAuth.

If OAuth server is available use below options to fetch access token from endpoint. For property description please refer to [Configuration](#) section.

```
spark.readStream.format("solace").option("host", "")
.option("vpn", "default")
.option("solace.apiProperties.AUTHENTICATION_SCHEME",
"AUTHENTICATION_SCHEME_OAUTH2")
.option("solace.oauth.client.auth-server-url", "")
.option("solace.oauth.client.client-id", "")
.option("solace.oauth.client.credentials.client-secret", "")
.option("solace.oauth.client.auth-server.client-certificate.file", "")
.option("solace.oauth.client.auth-server.truststore.file", "")
.option("solace.oauth.client.auth-server.truststore.password", "")
.option("solace.oauth.client.auth-server.ssl.validate-certificate", false)
.option("solace.oauth.client.token.refresh.interval", 110)
```

If rotating access token is present in file accessible by connector use below options to enable OAuth authentication to read access token from file. For property description please refer to [Solace OAuth](#)

**Client Refresh Interval** property. In case when access token file is not updated, connector retries the connection based on **Reconnect Retries** and stops if authentication is not successful.

```
spark.readStream.format("solace").option("host", "")
  .option("vpn", "")
  .option("solace.apiProperties.AUTHENTICATION_SCHEME",
"AUTHENTICATION_SCHEME_OAUTH2")
  .option("solace.oauth.client.access-token", "<absolute-path-to-token-file>")
  .option("solace.oauth.client.token.refresh.interval", 110)
```



When access token is read from file, it may lose some of its expiry time by the time it is accessed by connector. It is recommended to have minimal time difference between writing to file and access by the connector so that a valid new token is updated in solace session before expiry of old token.

Below is an example on how to use client certificate authentication when connecting to Solace.

```
sparkSession.readStream().format("solace")
  .option("host", "")
  .option("vpn", "default")
  .option("username", "")
  .option("solace.apiProperties.AUTHENTICATION_SCHEME",
"AUTHENTICATION_SCHEME_CLIENT_CERTIFICATE")
  .option("solace.apiProperties.SSL_TRUST_STORE", "<path-to-jks-file>")
  .option("solace.apiProperties.SSL_TRUST_STORE_FORMAT", "jks")
  .option("solace.apiProperties.SSL_TRUST_STORE_PASSWORD", "")
  .option("solace.apiProperties.SSL_KEY_STORE", "<path-to-jks-file>")
  .option("solace.apiProperties.SSL_KEY_STORE_FORMAT", "jks")
  .option("solace.apiProperties.SSL_KEY_STORE_PASSWORD", "")
```

For more properties please refer to [Solace Java API documentation for com.solacesystems.jcsmp.JCSMPPProperties](#)

## Using Databricks Secret Management

If Solace Spark Connector is deployed in Databricks, it is recommended to use Databricks secrets to store sensitive credentials.

To configure secrets refer to the [Databricks documentation](#).

You can reference those secrets in your Spark cluster using the same Spark config options:

Below is an example on how to retrieve username and password from Databricks secrets and connect to Solace.

```
spark.readStream.format("solace").option("host", dbutils.secrets.get(scope =
"solace-dev-credentials", key = "host"))
```

```

.option("vpn", "default")
.option("username", dbutils.secrets.get(scope = "solace-dev-credentials", key =
"username"))
.option("password", dbutils.secrets.get(scope = "solace-dev-credentials", key =
"password"))

```

OAuth based authentication to Solace using Databricks secrets. The certificates can be stored in cloud object storage, and you can restrict access to the certificates only to cluster that can access Solace. See [Data governance with Unity Catalog](#).

```

spark.readStream.format("solace").option("host", dbutils.secrets.get(scope =
"solace-dev-credentials", key = "host"))
.option("vpn", "default")
.option("solace.apiProperties.AUTHENTICATION_SCHEME",
"AUTHENTICATION_SCHEME_OAUTH2")
.option("solace.oauth.client.auth-server-url", "")
.option("solace.oauth.client.client-id", dbutils.secrets.get(scope = "solace-dev-
credentials", key = "client-id"))
.option("solace.oauth.client.credentials.client-secret", dbutils.secrets.get(scope
= "solace-dev-credentials", key = "client-secret"))
.option("solace.oauth.client.auth-server.client-certificate.file", "")
.option("solace.oauth.client.auth-server.truststore.file", "")
.option("solace.oauth.client.auth-server.truststore.password", dbutils.secrets.
get(scope = "solace-dev-credentials", key = "truststore-password"))
.option("solace.oauth.client.auth-server.ssl.validate-certificate", false)
.option("solace.oauth.client.token.refresh.interval", 110)

```

## Message Replay

Solace Spark Connector can replay messages using Solace Replay Log. Connector can replay all messages or after specific replication group message id or after specific timestamp. Please refer to [Message Replay Configuration](#) to enable replay log in Solace PubSub+ broker.

## Solace Spark Streaming Source Schema Structure

Solace Spark Connector transforms the incoming message to Spark row with below schema definition.

Column Name	Column Type	Description
Id	String	Represents Message ID present in message. This value is based on Offset_Indicator option. By, default it returns replication group message id.
Payload	Binary	Represents payload in binary format.

Column Name	Column Type	Description
PartitionKey	String	Represents Partition Key if present in message.
Topic	String	Represents the topic on which message is published.
TimeStamp	Timestamp	Represents sender timestamp if present in message. By default it returns the timestamp when message is received by connector.
Headers	Map<string, binary>	Represent message headers if present in message. This column is created only when includeHeaders option is set to true.

## Solace Spark Streaming Sink Schema Structure

Solace Spark Connector transforms the incoming message to Spark row with below schema definition.

Column Name	Column Type	Description
Id	String	Set the message id for the published message. This will be overwritten if message id is set using the id option. If no message id is set connector will throw an exception as message id is required to track the state of published messages. In case of publish failure the message id along with exception is logged.
Payload	Binary	Payload to be added to the published message. If no payload is set connector will throw an exception.
PartitionKey(Optional)	String	Partition Key for the published message. Useful when published message topic is subscribed by partitioned queues.
Topic(Optional)	String	Set the topic for the published message. This will be overwritten if topic is set using the topic option. If no topic is set connector will throw an exception as topic is required to publish a message.

Column Name	Column Type	Description
TimeStamp(Optional)	Timestamp	Set the timestamp for published message. This column is mapped to Sender Timestamp field in Solace Message.
Headers(Optional)	Map<string, binary>	Set the headers to be added in published message. This column is mapped only when includeHeaders option is set to true.

# Configuration

## Solace Spark Connector Source Configuration Options

Config Option	Type	Valid Values	Default Value	Description
host	string	any		Fully Qualified Solace Hostname with protocol and port number.
vpn	string	any		Solace VPN name.
username	String	any		Solace Client Username.
password	string	any		Solace Client Username password.
connectRetries	int	(-1) or greater	0	The number of times to attempt and retry a connection during initial connection setup. Zero means no automatic connection retries (that is, try once and give up). -1 means "retry forever".
reconnectRetries	int	(-1) or greater	3	The number of times to attempt to reconnect. Zero means no automatic reconnection retries (that is, try once and give up). -1 means "retry forever".

Config Option	Type	Valid Values	Default Value	Description
<code>connectRetriesPerHost</code>	int	(-1) or greater	0	When using a host list for the HOST property, this property defines how many times to try to connect or reconnect to a single host before moving to the next host in the list. NOTE: This property works in conjunction with the connect and reconnect retries settings; it does not replace them. Valid values are $\geq -1$ . 0 means make a single connection attempt (that is, 0 retries). -1 means attempt an infinite number of reconnect retries (that is, the API only tries to connect or reconnect to first host listed.)
<code>reconnectRetryWaitInMillis</code>	int	0 - 60000	3000	How much time in (MS) to wait between each attempt to connect or reconnect to a host. If connect or reconnect attempt to host is not successful, the API waits for the amount of time set for <code>reconnectRetryWaitInMillis</code> , and then makes another connect or reconnect attempt.

Config Option	Type	Valid Values	Default Value	Description
<code>solace.apiProperties.&lt;Property&gt;</code>	any	any		<p>Any additional Solace Java API properties can be set through configuring <code>solace.apiProperties.&lt;Property&gt;</code> where <code>&lt;Property&gt;</code> is the name of the property as defined in the <a href="#">Solace Java API documentation for <code>com.solacesystems.jcsmp.JCSMPPProperties</code></a>, for example:</p> <pre>solace.apiProperties .reapply_subscriptions=false solace.apiProperties .client_channel_properties.keepAliveIntervalInMillis=3000</pre>
<code>solace.oauth.client.access-token</code>	string	absolute file path to token file	empty	<p>Set this configuration, if rotating access token is present in file. In this case connector will read access token directly from file instead of sending request to OAuth Server. Please note <a href="#">Solace OAuth Client Refresh Interval</a> should be set to read access token from file at regular intervals.</p>
<code>solace.oauth.client.auth-server-url</code>	string	any	empty	<p>Full representation of token endpoint to fetch access token.</p>
<code>solace.oauth.client.client-id</code>	string	any	empty	<p>OAuth Client ID</p>
<code>solace.oauth.client.credentials.client-secret</code>	string	any	empty	<p>OAuth Client Secret</p>

Config Option	Type	Valid Values	Default Value	Description
<code>solace.oauth.client.auth-server.client-certificate.file</code>	string	any	empty	Absolute path to X.509 client certificate file for TLS connections. Make sure file path is accessible by the connector.
<code>solace.oauth.client.auth-server.truststore.file</code>	string	any	empty	<p>Absolute path to trust store file for TLS connections. This property works in two ways</p> <ol style="list-style-type: none"> <li>1. If JKS file is available in cluster configure absolute path so that connector will load the JKS file.</li> <li>2. If <code>solace.oauth.client.auth-server.client-certificate.file</code> is configured simply provide a path(should include file name as well). The connector will load the client certificate to key store and saves to JKS file .</li> </ol>
<code>solace.oauth.client.auth-server.truststore.password</code>	string	any	empty	<p>Password for JKS file. This property works in two ways</p> <ol style="list-style-type: none"> <li>1. If JKS file is available in cluster provide the password to JKS file.</li> <li>2. If <code>solace.oauth.client.auth-server.client-certificate.file</code> is configured simply provide a password which will be used to protect the JKS file created in above configuration option 2.</li> </ol>

Config Option	Type	Valid Values	Default Value	Description
<code>solace.oauth.client.auth-server.ssl.validate-certificate</code>	boolean	any	true	Boolean value to enable or disable ssl certificate validation. If set to false connector will send TLS request without any validation.
<code>solace.oauth.client.auth-server.tls.version</code>	string	SSL, TLS, TLSv1, TLSv1.1, TLSv1.2, TLSv1.3	TLSv1.2	Indicate the type of SSL connection.
<code>solace.oauth.client.token.refresh.interval</code>	integer	positive integer value	60	Interval(Seconds) to fetch access token by the connector to avoid disconnection on token expiry. This value should be less than your token expiry time.
<code>solace.oauth.client.token.fetch.timeout</code>	integer	positive integer value	100	Connection timeout(MS) for access token request.
<code>queue</code>	string	any		Solace Queue name.
<code>batchSize</code>	int	any	1	Set number of messages to be processed in batch. The connector can stream data in batches to Spark based on configured size.
<code>replayStrategy</code>	string	BEGINNING   TIMEBASED   REPLICATION-GROUP-MESSAGE-ID	empty	Set the replay strategy if messages need to be replayed from broker to connector. For more information refer to <a href="#">SolaceReplayConfiguration</a>
<code>replayReplicationGroupMessageId</code>	string	valid-replication-group-message-id	empty	Set the property if replay strategy is REPLICATION-GROUP-MESSAGE-ID. Message playback is started after this replication group message id.

Config Option	Type	Valid Values	Default Value	Description
<code>replayStartTime</code>	string	datetime string<yyyy-MM-ddT HH:mm:ss>	empty	Set the property if replay strategy is TIMEBASED. Any messages in the replay log equal to, or newer than, the specified date and time that match the endpoint's subscriptions are replayed to the connector. The date can't be earlier than the date the replay log was created, otherwise replay will fail.
<code>replayStartTimeTimezone</code>	string	valid timezone	UTC	Set the property if replay strategy is TIMEBASED.
<code>ackLastProcessedMessages</code>	boolean	true or false	false	Set this value to true if connector needs to identify and acknowledge processed messages in last run during restarts. The connector purely depends on checkpoint generated during Spark commit. We recommended enabling this configuration only when your downstream system has processed data in previous run.  <div style="display: flex; align-items: center;">  <p>This property will be void if replay strategy is enabled.</p> </div>

Config Option	Type	Valid Values	Default Value	Description
offsetIndicator	string	any	MESSAGE_ID, CORRELATION_ID, APPLICATION_MESSAGE_ID, <CUSTOM_USER_PROPERTY>	<p>Set this value if your Solace Message has unique ID in message header. Supported Values are</p> <ul style="list-style-type: none"> <li>• MESSAGE_ID</li> <li>• CORRELATION_ID</li> <li>• APPLICATION_MESSAGE_ID</li> <li>• &lt;CUSTOM_USER_PROPERTY&gt; - refers to one of headers in user properties Header.</li> </ul> <p>Note: Default value uses replication group message ID property as offset indicator. ReplicationGroupMessageId is a unique message id across a broker cluster.</p>
includeHeaders	boolean	true or false	false	Set this value to true if message headers need to be included in output.
partitions	int	any	1	Sets the number of consumers for configured queue. If more the one worker node is present, consumers are split across worker nodes for efficient processing.

Config Option	Type	Valid Values	Default Value	Description
<code>createFlowsOnSameSession(deprecated)</code>	boolean	true or false	false	If enabled consumer flows are enabled on same session. The number of consumer flows is equal to number of partitions configured. This is helpful when users want to optimize on number of connections created from Spark. By default, the connector creates a new connection for each consumer.
<code>lvq.name</code>	string	valid lvq name configured on solace broker	empty	Set the name of LVQ configured on solace broker. This is required to communicate checkpoint information from worker node to driver node and it also ensures that checkpoint information is present in both LVQ and Spark checkpoint directory
<code>lvq.topic</code>	string	valid solace topic	empty	Set the name of the topic subscribed by LVQ on solace broker. Each worker node publishes checkpoint information to LVQ on this topic.

## Solace Spark Connector Sink Configuration Options

Config Option	Type	Valid Values	Default Value	Description
<code>host</code>	string	any		Fully Qualified Solace Hostname with protocol and port number.
<code>vpn</code>	string	any		Solace VPN name.
<code>username</code>	String	any		Solace Client Username.
<code>password</code>	string	any		Solace Client Username password.

Config Option	Type	Valid Values	Default Value	Description
<code>connectRetries</code>	int	(-1) or greater	0	The number of times to attempt and retry a connection during initial connection setup. Zero means no automatic connection retries (that is, try once and give up). -1 means "retry forever".
<code>reconnectRetries</code>	int	(-1) or greater	3	The number of times to attempt to reconnect. Zero means no automatic reconnection retries (that is, try once and give up). -1 means "retry forever".
<code>connectRetriesPerHost</code>	int	(-1) or greater	0	When using a host list for the HOST property, this property defines how many times to try to connect or reconnect to a single host before moving to the next host in the list. NOTE: This property works in conjunction with the connect and reconnect retries settings; it does not replace them. Valid values are $\geq -1$ . 0 means make a single connection attempt (that is, 0 retries). -1 means attempt an infinite number of reconnect retries (that is, the API only tries to connect or reconnect to first host listed.)

Config Option	Type	Valid Values	Default Value	Description
<code>reconnectRetryWaitInMillis</code>	int	0 - 60000	3000	How much time in (MS) to wait between each attempt to connect or reconnect to a host. If a connect or reconnect attempt to host is not successful, the API waits for the amount of time set for <code>reconnectRetryWaitInMillis</code> , and then makes another connect or reconnect attempt.
<code>solace.apiProperties.&lt;Property&gt;</code>	any	any		Any additional Solace Java API properties can be set through configuring <code>solace.apiProperties.&lt;Property&gt;</code> where <code>&lt;Property&gt;</code> is the name of the property as defined in the <a href="#">Solace Java API documentation for <code>com.solacesystems.jcsmp.JCSMPPProperties</code></a> , for example: <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <pre>solace.apiProperties .pub_ack_window_size =50</pre> </div>
<code>solace.oauth.client.access-token</code>	string	absolute file path to token file	empty	Set this configuration, if rotating access token is present in file. In this case connector will read access token directly from file instead of sending request to OAuth Server. Please note <a href="#">Solace OAuth Client Refresh Interval</a> should be set to read access token from file at regular intervals.
<code>solace.oauth.client.auth-server-url</code>	string	any	empty	Full representation of token endpoint to fetch access token.

Config Option	Type	Valid Values	Default Value	Description
<code>solace.oauth.client.client-id</code>	string	any	empty	OAuth Client ID
<code>solace.oauth.client.credentials.client-secret</code>	string	any	empty	OAuth Client Secret
<code>solace.oauth.client.auth-server.client-certificate.file</code>	string	any	empty	Absolute path to X.509 client certificate file for TLS connections. Make sure file path is accessible by the connector.
<code>solace.oauth.client.auth-server.truststore.file</code>	string	any	empty	<p>Absolute path to trust store file for TLS connections. This property works in two ways</p> <ol style="list-style-type: none"> <li>1. If JKS file is available in cluster configure absolute path so that connector will load the JKS file.</li> <li>2. If <code>solace.oauth.client.auth-server.client-certificate.file</code> is configured simply provide a path(should include file name as well). The connector will load the client certificate to key store and saves to JKS file .</li> </ol>
<code>solace.oauth.client.auth-server.truststore.password</code>	string	any	empty	<p>Password for JKS file. This property works in two ways</p> <ol style="list-style-type: none"> <li>1. If JKS file is available in cluster provide the password to JKS file.</li> <li>2. If <code>solace.oauth.client.auth-server.client-certificate.file</code> is configured simply provide a password which will be used to protect the JKS file created in above configuration option 2.</li> </ol>

Config Option	Type	Valid Values	Default Value	Description
<code>solace.oauth.client.auth-server.ssl.validate-certificate</code>	boolean	any	true	Boolean value to enable or disable ssl certificate validation. If set to false connector will send TLS request without any validation.
<code>solace.oauth.client.auth-server.tls.version</code>	string	SSL, TLS, TLSv1, TLSv1.1, TLSv1.2, TLSv1.3	TLSv1.2	Indicate the type of SSL connection.
<code>solace.oauth.client.token.refresh.interval</code>	integer	positive integer value	60	Interval(Seconds) to fetch access token by the connector to avoid disconnection on token expiry. This value should be less than your token expiry time.
<code>solace.oauth.client.token.fetch.timeout</code>	integer	positive integer value	100	Connection timeout(MS) for access token request.
<code>topic</code>	string	any		Sets the topic that all rows will be published to Solace. This option overrides any topic column that may exist in the data.
<code>id</code>	string	any		Sets the message id to all the messages published to Solace. This option overrides any Id column that may exist in the data.
<code>batchSize</code>	int	any	1	Set number of messages to be processed in batch. This should be set to <code>dataframe.count()</code> .
<code>includeHeaders</code>	boolean	true or false	false	Set this value to true if message headers in the row need to be added to published message.



This connector is tested on Databricks Cluster Version 15.4 LTS (includes Apache Spark 3.5.0, Scala 2.12) with photon acceleration disabled.

# License

This project is licensed under the Solace Community License, Version 1.0. - See the [LICENSE](#) file for details.

# Support

Support is offered best effort via our [Solace Developer Community](#).

Premium support options are available, please [Contact Solace](#).