pubsubplus-connector-debezium

Table of Contents

Getting Started

Solace

Solace Google PubSub Spring Cloud stream connector enables reading of data from Google PubSub and publishing it to Solace. Binders for Google PubSub and Solace are used to consume/publish the data.

Free Solace Access

- Sign up for a free Solace Cloud service
- Download the free feature-complete Standard Edition software broker
- Quickstart Video for Solace PubSub+ Docker container

Setup Solace PubSub+ Broker

This broker can be a part of a Solace event mesh which has many interconnected Solace Brokers.

Access to a Solace messaging service, Solace PubSub+, can be achieved in either one of the three flavours

1. Hardware Appliance
2. Software broker image (Docker, Virtual image)
3. Solace Cloud service instance

This tutorial will walk you through setting up a Solace Cloud service instance, which also gives you access to the Event Portal. If you are interested in setting up a local broker running on Docker or a virtual machine check out the PubSub+ Event Broker: Software documentation

Sign up for free Solace Cloud account

Navigate to the Create a New Account page and fill out the required information. No credit card required!

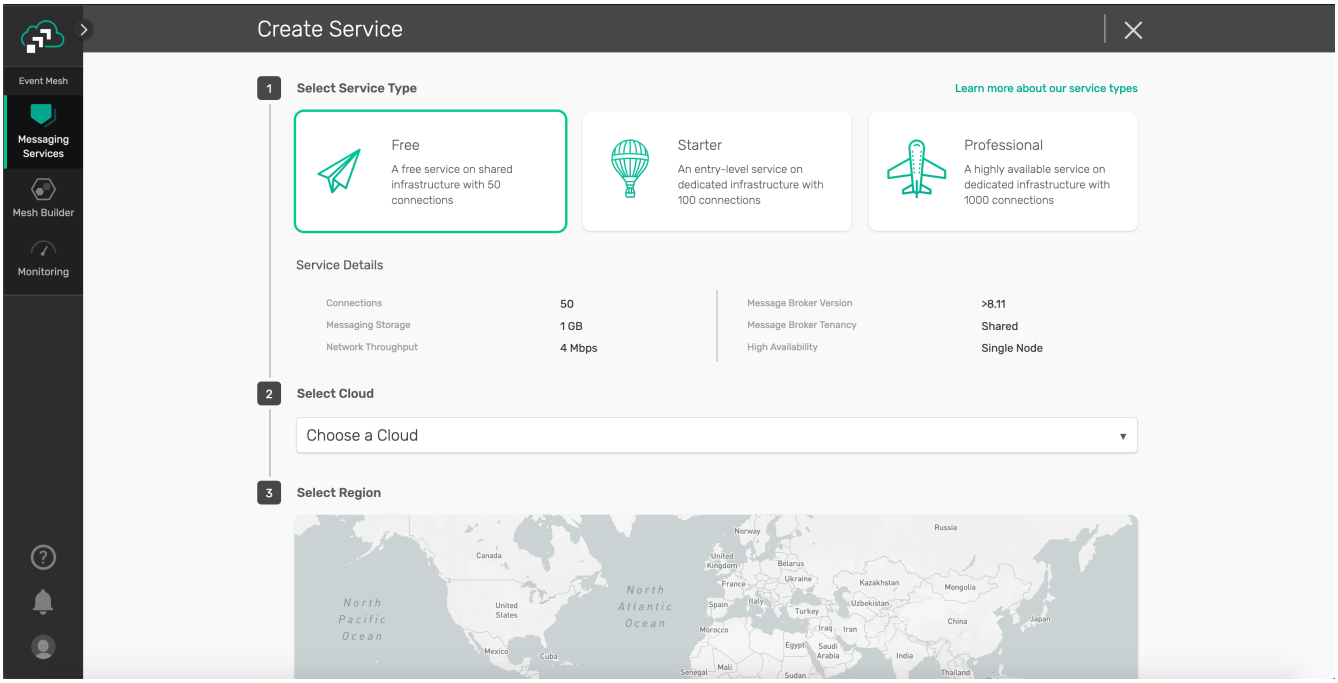
Create a messaging service

After you create your Solace Cloud account and sign in to the Solace Cloud Console, you'll be routed to the event mesh page.

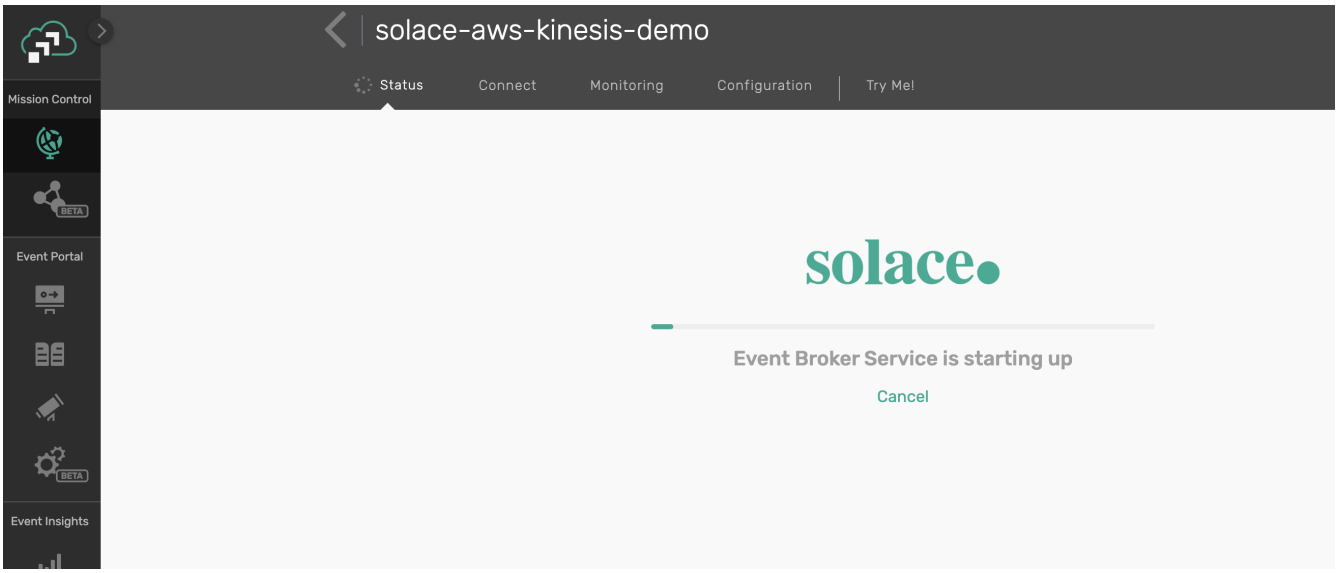
[solace setup1] | */home/runner/work/pubsubplus-connector-debezium/pubsubplus-connector-debezium/src/docs/asciidoc/libs/../../images/solace_setup1.webp*

Click on 'Cluster Manager' and all the messaging services associated with your account will show up if you have any already created. To create a new service, click either button as depicted in the image below:

Fill out all the details for your messaging service, and then click "Create" at the bottom of the page.



Your service should be ready to use in a few minutes!



CDC Debezium

CDC Debezium is based on Spring Cloud Stream architecture. It uses the Debezium embedded engine to connect to the configured database and process change events. Visit this link for more information.

Architecture

This connector is based on Spring Cloud Stream architecture and Solace Connector Framework. Solace Connector framework is built on Spring Java and it is the core component to manage to connector lifecycle. Solace Connector Framework provides the ability to configure upto 20 workflows(Each workflow represents data flow from One Source to One Destination) and enables the connector to configure in High Availability mode. Apart from this the framework handles logging and exporting connector metrics to different sources. The connector uses Debezium

Embedded Engine provided by CDC Debezium library to communicate with configured database

[debezium cdc architecture] | */home/runner/work/pubsubplus-connector-debezium/pubsubplus-connector-debezium/src/docs/asciidoc/libs/../images/debezium-cdc-architecture.png*

Features

CDC Debezium supported Databases

CDC Debezium supports connecting to five datastores: `MySQL, PostgreSQL, MongoDB, Oracle and SQL Server database`. Currently this connector is tested with MySQL, Postgres and MongoDB database and single workflow. Users can configure multiple tables in a single workflow to be processed by the connector.

Offset Management

CDC Debezium supports the following to store offsets`(Memory, File, Kafka, Metadata). `This connector by default uses `File` to store offsets. Users can configure the file location that is accessible by the connector. CDC Debezium reads information from the source and periodically records offsets that define how much information is processed. Should the connector be restarted, it will use the last recorded offset to know where in the source information it should resume reading.

Dynamic Destination

Users can configure `"dynamicDestination"` property to build dynamic topic based on table and column data. For example `orders:spring/cdc/mysql/{tablename}/APAC/{col|order_id};users:spring/cdc/mysql/{tablename}/{col|user_id}`.

So orders table topic will be constructed as `spring/cdc/mysql/orders/APAC/ORD_12` and same convention follows for users table `spring/cdc/mysql/users/USR_12`

If column names are incorrect connector will fail if failIfDynamicDestinationMismatch is set to true or fallback to output destination(static topic) if failIfDynamicDestinationMismatch is set to false.

Debezium captures DDL statements as part of CDC events. In general when dynamic destination is configured and failIfDynamicDestinationMismatch is set to false DDL events are published to static topic. In cases where DDL events are not required by down stream systems, this configuration will be helpful

Exception Handling

CDC debezium process data in batches by default. In order to handle exceptions the connector provides a configuration where failed message can be reprocessed basing on retry configuration. Once it reaches the limit, the connector sends a stop signal CDC Debezium which stops the Debezium Embedded engine. In this case the entire batch will not be written to offset storage and will be reprocessed after resolving the exception. During reprocesses the consumer of change events may see duplicates.

Handling duplicates

CDC debezium provides two configurations `max.batch.size` and `max.queue.size` which can be used to control the number of messages handled in a batch.

`max.batch.size` defines the number of records to be processed in batch. Setting this to an optimal value will reduce the number of duplicates in case of batch reprocessing. Set to 1 to avoid duplicates. Default value of `max.batch.size is 2048`

`max.queue.size` defines the number of records that debezium can hold in its internal queue after reading from database. This value should always be greater than max.batch.size. Default value of `max.queue.size is 8192`

Decimal Handling Mode

This version addresses the issue of representing numerical/decimal data types in actual format instead of Base64 string. However, in case of Float or Real type the representation is still Base64 string. In order to resolve this we recommend setting decimal.handling.mode to double or string to get actual value. Below table demonstrates how values are represented for float and real data types. For more details on this configuration refer to User-Guide for respective database.

Config Option	Value in Database	Value in CDC event
`decimal.handling.mode=double`	`90`	`90.0`
`decimal.handling.mode=string`	`90`	`90`

Converting Base64 string to actual value with default decimal.handling.mode = precise will have an impact on performance since the connector need to traverse and update entire payload before emitting the CDC event. This conversion will be handled in future releases.

MongoDB Nested Objects/Arrays

MongoDB supports hierarchical documents where there can be nested objects or arrays. Dynamic Destination feature can not be supported in hierarchical documents. However, properties at root level can be configured in dynamic destination.

Other Features & Configuration

Please refer to User-Guide.pdf available in dist/pubsubplus-connector-debezium-1.0.0-SNAPSHOT/

Java dependencies in the `/libs` folder

Preface

This `/libs` directory is provided as a default location for the Java library dependencies (external `jar` files) that are either required or required only when using certain features of the connector (such

as Prometheus libraries when using the metrics export to Prometheus feature in your connector configuration).

Solace does not provide the required JAR files due to licensing considerations. These JAR files are required as part of the deployment of the connector for it to operate correctly.

Micrometer Metrics exporter dependencies

The connector makes use of Micrometer.io to provide easy, configuration-driven exporting of connector metrics to many 3rd-party metrics monitoring solutions (see the full list on the Micrometer site).

These services can be configured in the connector configuration (there are commented in the sample configuration provided to guide you), but must have the necessary Java dependencies in the `/libs` directory to operate.

The JAR files that are necessary depend on which metrics service you are configuring. The details for each provider can be found in the Micrometer documentation. For example, for Prometheus, the Micrometer instructions list `io.micrometer.micrometer-registry-prometheus` as the main client library required and the Maven Central page for this library provides the download links for the main `jar` and its compile dependencies. These JAR files are placed in the connector's `/libs` directory.