pubsubplus-connector-database

User Guide

Solace Corporation

Version 1.4.0

solace.

Table of Contents

Preface

Solace Database Connector

Getting Started

Assuming you're using the default `application.yml` within this package, following one of the below quick start guides will result in a connector that will connect to the PubSub+ broker and SolaceDBConnector using default credentials, with 2 workflows enabled, workflow 0 and workflow 1. Where:

- Workflow 0 is consuming messages from the Solace PubSub+ queue, `Solace/Queue/0`, and publishing them to the SolaceDBConnector producer destination, `producer-destination`.
- Workflow 1 is consuming messages from the SolaceDBConnector consumer destination, `consumer-destination`, and publishing them to the Solace PubSub+ topic, `Solace/Topic/1`.

A workflow is the configuration of a flow of messages from a source to a target. The connector supports up to 20 concurrent workflows per instance.

The connector will not provision queues which do not exist.

Prerequisites

- Solace PubSub+ Event Broker
- SolaceDBConnector

Quick Start common steps

These are the steps that are required to run all quick-start examples:

1. Update the provided `samples/config/application.yml` with the values for your deployment.

Quick Start: Running the connector via command line

Run:

```
java -jar pubsubplus-connector-database-1.4.0.jar --spring.config.additional-location
=file:samples/config/
```

By default, this command will detect any Spring Boot config files as per Spring Boot's default locations.

For more info, see Configure Locations to Find Spring Property Files.

Quick Start: Running the connector via `start.sh` script

For convenience purposes, connector may be also started through the shell script using following syntax:

```
./bin/start.sh [-l FOLDER] [-p PROFILE] [-c FOLDER] [-j FILE] [-o OPTIONS] [-b]
```

In case of invalid parameters:

```
./bin/start.sh -l dummy_folder -c dummy_folder -j dummy_file.jar
```

the script shows you all errors at once:

```
pubsubplus-connector-database

Connector startup failed:

Following folder doesn't exists on your filesystem:     'dummy_folder'
Following folder doesn't exists on your filesystem:     'dummy_folder'
Following file doesn't exists on your filesystem:       'dummy_file.jar'
```

In case of missing parameters, script will run with predefined values, which are the following:

Parameter	Default Value	Description
`-l, --libs`	`./libs`	Directory containing required and optional dependency jars. Such as Micrometer metrics export dependencies (if configured). If not specified, it will use the current `./libs/` folder.
`-p, --profile`	`empty, no profile is used`	Profile to be used with connector's configuration. The configuration file named 'application-<profile>.yml' will be used. Default value is set to use no profile.
`-c, --config`	`current folder`	Path to the folder containing the configuration files to be applied during connector startup for chosen profile. By default it is set to the current folder.
`-j, --jar`	`pubsubplus-connector-database-1.4.0.jar`	Path to specified JAR file to start connector. If not specified, the default jar file is used from the current folder.

Parameter	Default Value	Description
-o, --options	no default values	Specifies JVM options used on Connector start.
-b, --background	N/A	Runs Connector in the background. No logs will be displayed and Connector continues running in detached mode
-h, --help	N/A	Prints some help information and exits

Script also provides some help information from command line:

```
pubsubplus-connector-database

Usage: start.sh [-l FOLDER] [-p PROFILE] [-c FOLDER] [-j FILE] [-o OPTIONS] [-b]

To start connector use following parameters:

-l | --libs        Directory containing required and optional
                   dependency jars. Such as Micrometer metrics
                   export dependencies (if configured).
                   If not specified, it will use the current
                   './libs/' folder

-p | --profile     Profile to be used with connector's configuration.
                   The configuration file named
                   'application-<profile>.yml' will be used
                   Default value is empty, no profile is used.

-c | --config      Path to the folder containing the
                   configuration files to be applied during
                   connector startup for chosen profile.
                   By default it is set to the current folder.

-j | --jar         Path to specified JAR file to start connector.
                   if not specified, the default jar file is used,
                   from the current folder

-o | --options     Specifies JVM options used on Connector start.
                   example: '-Xms64M -Xmx1G'

-b | --background  Runs the Connector in the background. No logs will be
                   displayed and Connector continues running in detached
                   mode.

-h | --help        Print this help page and exit
```

Quick Start: Running the connector as a Container

A sample docker compose file has been packaged for your convenience:

1. Change to the `docker` directory:

   ```
   cd samples/docker
   ```

   This directory contains both the `docker-compose.yml` file as well as a `.env` file that contains environment secrets required for the container's health-check.

2. Run the connector:

   ```
   docker-compose up -d
   ```

   This sample docker compose file will:

   - Expose the connector's `8090` web port to `8090` on the host.
   - Connect to PubSub+ and SolaceDBConnector exposed on the host using default ports.
   - Mount the `samples/config` directory.
   - Mount the previously defined `libs` directory.
   - Create a `healthcheck` user with read-only permissions.
     - The default username and password for this user can be found within the `.env` file.
     - This will override any users you have defined in your `application.yml`. See here for more info.
   - Uses the connector's management health endpoint as the container's healthcheck.

For more info about how to use and configure this container, see the connector's container documentation.

Enabling Workflows

The provided `application.yml` enables workflow 0 and 1. To enable additional workflows, define the following properties in the `application.yml`, where `<workflow-id>` is a value between `[0-19]`:

```
spring:
  cloud:
    stream:
      bindings: # Workflow bindings
        input-<workflow-id>:
          destination: <input-destination> # Queue name
          binder: (solace|solace-db) # Input system
        output-<workflow-id>:
          destination: <output-destination> # Topic name
          binder: (solace|solace-db) # Output system

solace:
  connector:
    workflows:
      <workflow-id>:
        enabled: true
```

The connector only supports workflows in the directions of:

- `solace` → `solace-db`
- `solace-db` → `solace`

For more info about Spring Cloud Stream and the Solace PubSub+ binder:

- Spring Cloud Stream Reference Guide
- Spring Cloud Stream Binder for Solace PubSub+

Configuring Connection Details

Solace PubSub+ Connection Details

The Spring Cloud Stream Binder for Solace PubSub+ uses Spring Boot Auto-Configuration for the Solace Java API to configure its session.

In the `application.yml`, this typically would be configured as follows:

```
solace:
  java:
    host: tcp://localhost:55555
    msg-vpn: default
    client-username: default
    client-password: default
```

For more info and options to configure the PubSub+ session, see Spring Boot Auto-Configuration for the Solace Java API.

Preventing Message Loss when Publishing to Topic-to-Queue Mappings

If the connector is publishing to a topic that is subscribed to by a queue, messages may be lost if they are rejected (e.g. if queue ingress is shutdown).

To prevent message loss, configure `reject-msg-to-sender-on-discard` with the `including-when-shutdown` flag.

Connecting to Multiple Systems

To connect to multiple systems of a same type, use the multiple binder syntax.

For instance:

```
spring:
  cloud:
    stream:
      binders:

        # 1st solace binder in this example
        solace1:
          type: solace
          environment:
            solace:
              java:
                host: tcp://localhost:55555

        # 2nd solace binder in this example
```

```
        solace2:
          type: solace
          environment:
            solace:
              java:
                host: tcp://other-host:55555

        # The only solace-db binder
        solace-db1:
          type: solace-db
          # Add `environment` property map here if you need to customize this binder.
          # But for this example, we'll assume that defaults are used.

        # Required for internal use
        undefined:
          type: undefined
      bindings:
        input-0:
          destination: <input-destination>
          binder: solace-db1
        output-0:
          destination: <output-destination>
          binder: solace1 # Reference 1st solace binder
        input-1:
          destination: <input-destination>
          binder: solace-db1
        output-1:
          destination: <output-destination>
          binder: solace2 # Reference 2nd solace binder
```

Defines two binders of type `solace` and one binder of type `solace-db` which are then referenced within bindings.

Note that each binder is configured independently under `spring.cloud.stream.binders.<binder-name>.environment`.

When connecting to multiple systems, all binder configuration must be specified using the multiple binder syntax for all binders.

Do not use single-binder configuration (e.g. `solace.java.*` at the root of your `application.yml`) while using the multiple binder syntax.

User-configured Header Transforms

Generally, the consumed message's headers are propagated through the connector to the output message. If you want to transform the headers, then you may do so as follows:

```
# <workflow-id> : The workflow ID ([0-19])
# <header> : The key for the outbound header
# <expression> : A SpEL expression which has "headers" as parameters

solace.connector.workflows.<workflow-id>.transform-
headers.expressions.<header>=<expression>
```

Example 1: To create a new header, `new_header`, for workflow `0` that is derived from the headers `foo` & `bar`:

```
solace.connector.workflows.0.transform-headers.expressions.new_header
="T(String).format('%s/abc/%s', headers.foo, headers.bar)"
```

Example 2: To remove the header, `delete_me`, for workflow `0`, set the header transform expression to `null`:

```
solace.connector.workflows.0.transform-headers.expressions.delete_me="null"
```

For more info about Spring Expression Language (SpEL) expressions:

- Spring Expression Language (SpEL)

User-configured Payload Transforms

Message payloads going through a workflow can be transformed using a Spring Expression Language (SpEL) expression as follows:

```
# <workflow-id> : The workflow ID ([0-19])
# <expression> : A SpEL expression

solace.connector.workflows.<workflow-id>.transform-payloads.expressions[0].transform
=<expression>
```

A SpEL expression may reference:

- `payload`: to access the message payload
- `headers.<header_name>`: to access a message header value
- Registered functions

While the syntax uses an array of expressions, only a single transform expression is supported in this release. Multiple transform expressions may be supported in future versions.

Registered Functions

Registered functions are built-in and can be called directly from SpEL expressions. To call a registered function, use the `#` character followed by the function name. The following table describes available registered functions:

Registered Function Signature	Description
`boolean isPayloadBytes(Object obj)`	Returns whether the object `obj` is an instanceof `byte[]` or not. Sample usage of this function within a SpEL expression: `"#isPayloadBytes(payload) ? true : false"`

Example 1: To normalize byte[] and String payloads as upper-cased String payloads or leave payloads unchanged when of different types.

```
solace.connector.workflows.0.transform-payloads.expressions[0].transform
="#isPayloadBytes(payload) ? new String(payload).toUpperCase() : payload instanceof
T(String) ? payload.toUpperCase() : payload"
```

Example 2: To convert String payloads to byte[] payloads using a charset retrieved from a message header or leave payloads unchanged when of different types.

```
solace.connector.workflows.0.transform-payloads.expressions[0].transform="payload
instanceof T(String) ?
payload.getBytes(T(java.nio.charset.Charset).forName(headers.charset)) : payload"
```

For more info about Spring Expression Language (SpEL) expressions:

- Spring Expression Language (SpEL)

Message Headers

Solace and solace-db headers can be created or manipulated using the User-configured Header Transforms feature described above.

Solace Headers

Solace headers exposed to the connector are documented within the Spring Cloud Stream Binder for Solace PubSub+ documentation.

Reserved Message Headers

The following are reserved header spaces:

- `solace_`
- `scst_`
- Any headers defined by the core Spring messaging framework. See Spring Integration: Message Headers for more info.

Any headers with these prefixes, that are not defined by the connector or any technology used by the connector, may not be backwards compatible in a future version of the connector.

Management and Monitoring Connector

Monitoring Connector's States

The Connector provides an ability to monitor its internal states through exposed endpoints provided by Spring Boot Actuator.

Actuator shares information through the endpoints reachable over HTTP(s). What endpoints are available is configured in the connector configuration file:

```
management:
  metrics:
    export:
      simple:
        enabled: true
  endpoints:
    web:
      exposure:
        include:
"health,metrics,loggers,logfile,channels,env,workflows,leaderelection,bindings"
```

The above example configuration enables metrics collection through the configuration parameter of `management.metrics.export.simple.enabled` set to `true` and then shares them through the HTTP(s) endpoint together with other sections configured for the current Connector.

The set of endpoints exposed through the HTTP(s) endpoint. Exposed endpoints are available in the connector UI and are also available to the PubSub+ Connector Manager. The operator may choose to not expose all or some of these endpoints. Endpoints not exposed will not be available in the connector web UI nor the PubSub+ Connector Manager.

The simple metrics registry is only to be used for testing. It is not a production-ready means of collecting metrics. In production, use a dedicated monitoring system (e.g. Datadog, Prometheus, etc) to collect metrics.

The Actuator endpoint now contains information about Connector's internal states shared over the following HTTP(s) endpoint:

```
GET: /actuator/
```

Here is the example of the data shared with the configuration above:

```
{
  "_links": {
    "self": {
      "href": "/actuator",
      "templated": false
```

```
    },
    "workflows": {
      "href": "/actuator/workflows",
      "templated": false
    },
    "workflows-workflowId": {
      "href": "/actuator/workflows/{workflowId}",
      "templated": true
    },
    "leaderelection": {
      "href": "/actuator/leaderelection",
      "templated": false
    },
    "health-path": {
      "href": "/actuator/health/{*path}",
      "templated": true
    },
    "health": {
      "href": "/actuator/health",
      "templated": false
    },
    "metrics": {
      "href": "/actuator/metrics",
      "templated": false
    },
    "metrics-requiredMetricName": {
      "href": "/actuator/metrics/{requiredMetricName}",
      "templated": true
    }
  }
}
```

Health

The connector reports its health status via the Spring Boot Actuator health endpoint.

To configure the information returned by the `health` endpoint, configure the following properties: `management.endpoint.health.show-details` and `management.endpoint.health.show-components`. Refer to Spring Boot documentation for details.

Health for the workflow, Solace binder, and solace-db binder components are exposed when `management.endpoint.health.show-components` is enabled. For example:

```
management:
  endpoint:
   health:
    show-components: always
    show-details: always
```

This would always show the full detail of the health check including the workflows and binders. The default value is `never`.

Workflow Health

A `workflows` health indicator is provided to show the health status for each of a connector's workflows. This health indicator has the following form:

```
{
  "status": "(UP|DOWN)",
  "components": {
    "<workflow-id>": {
      "status": "(UP|DOWN)",
      "details": {
        "error": "<error message>"
      }
    }
  }
}
```

Health Status	Description
`UP`	Status indicating that the workflow is functioning as expected.
`DOWN`	Status indicating that the workflow is unhealthy. User intervention may be required.

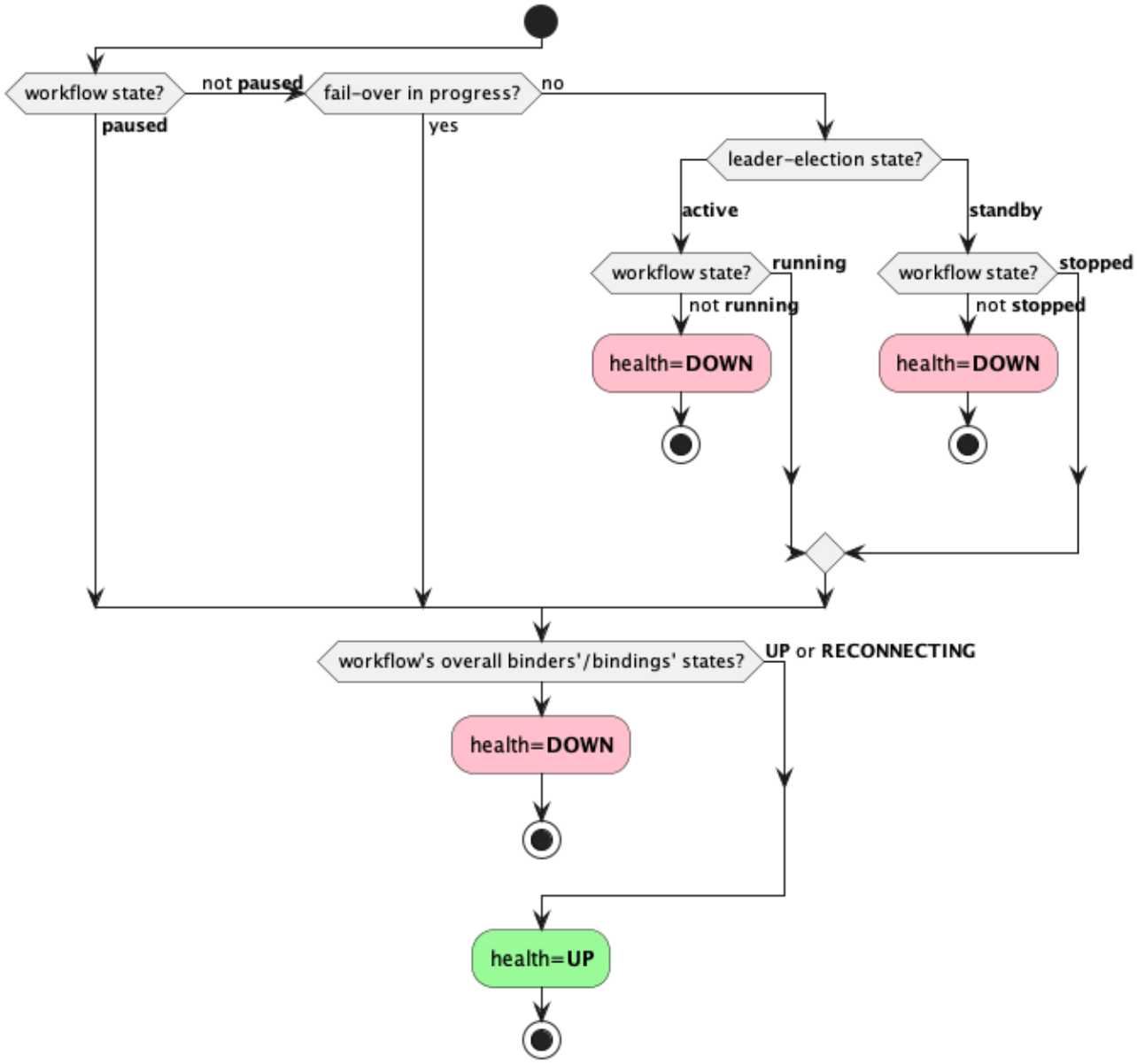


Figure 1. Workflow Health Resolution Diagram

This health indicator is enabled default. To disable it, add this to your configuration:

```
management.health.workflows.enabled=false
```

Solace Binder Health

For details, see the Solace binder documentation.

Leader Election

The connector has 3 leader election modes:

Leader Election Mode	Description
Standalone (Default)	A single instance of a connector without any leader election capabilities.
Active-Active	A participant in a cluster of connector instances where all instances are active.
Active-Standby	A participant in a cluster of connector instances where only one instance is active (i.e. the leader), and the others are standby.

Operators can configure the leader election mode by setting:

```
solace.connector.management.leader-election.mode
=(standalone|active_active|active_standby)
```

Leader Election Modes: Standalone / Active-Active

All enabled workflows are started during connector startup and the connector is considered as always active.

Leader Election Mode: Active-Standby

If the connector is in active-standby mode, then a PubSub+ management session and management queue must be configured as follows:

```
solace.connector.leader-election.mode=active_standby

# Management session
# Exact same interface as solace.java.*
solace.connector.management.session.host=<management-host>
solace.connector.management.session.msgVpn=<management-vpn>
solace.connector.management.session.client-username=<client-username>
solace.connector.management.session.client-password=<client-password>
solace.connector.management.session.<other-property-name>=<value>

# Management queue name accessible by the management session
# Must have exclusive access type
solace.connector.management.queue=<management-queue-name>
```

To determine if the connector is `active` or `standby`, it will create a flow to the management queue. If this flow is active, then the connector's state is `active` and will start its enabled workflows. Otherwise, if this flow is inactive, then the connector's state is `standby` and will stop its enabled workflows.

At a macro level for a cluster of connectors, fail-over only happens when there are infrastructure failures (e.g. JVM goes down or networking failures to the management queue).

If a workflow fails to be started/stopped during fail-over, it will retry up to some maximum defined by the config option, `solace.connector.management.leader-election.fail-over.max-attempts`.

During fail-over, the connector will attempt to start/stop all enabled workflows. Once an attempt has been made to start/stop each workflow, then the connector has transitioned to active/standby mode regardless of the status of the workflows.

Leader Election Management Endpoint

A custom `leaderelection` management endpoint was provided using Spring Actuator.

Navigate to the connector's `leaderelection` management endpoint to view its leader election status.

Endpoint	Operation	Payloads
`/leaderelection`	Read (HTTP `GET`)	**Request:** None. **Response:** `{ "state": "(active|standby)", "mode": "(standalone|active_active|active_standby)" }`

Workflow Management

Workflow Management Endpoint

A custom `workflows` management endpoint using Spring Actuator was provided to manage workflows.

To enable the `workflows` management endpoint:

```
management:
  endpoints:
    web:
      exposure:
        include: 'workflows'
```

Once the `workflows` management endpoint is enabled, the following operations can be performed:

Endpoint	Operation	Payloads
`/workflows`	Read (HTTP `GET`)	**Request:** None. **Response:** Same payload as the `/workflows/{workflowId}` read operation, but as a list of all workflows.
`/workflows/{workflowId}`	Read (HTTP `GET`)	**Request:** None. **Response:** `{ "id": "<workflowId>", "enabled": (true|false), "state": "(running|stopped|paused|unknown)", "inputBindings": [ "<input-binding>" ], "outputBindings": [ "<output-binding>" ] }`

Endpoint	Operation	Payloads
`/workflows/{workflowId}`	Write (HTTP `POST`)	**Request:** `{ "state": "STARTED|STOPPED|PAUSED|RESUMED" }` **Response:** None.

Only workflows with Solace PubSub+ consumers (where the `solace` binder is defined in the `input-#`) support pause/resume.

Some features require for the connector to manage workflow lifecycles. There's no guarantee that workflow states will persist when write operations are used to change workflow states while such features are in use.

For example: When the connector is configured in the active-standby leader election mode, workflows will automatically transition from `running` to `stopped` when the connector fails over from `active` to `standby`. Vice versa for a fail-over in the opposite direction.

Workflow States

A workflow's state is defined as the aggregate states of its bindings (see the `bindings` management endpoint) as follows:

Workflow State	Condition
`running`	All bindings have `state="running"`.
`stopped`	All bindings have `state="stopped"`.
`paused`	All consumer bindings and all pausable producer bindings have `state="paused"`.
`unknown`	None of the other states. Represents an inconsistent aggregate binding state.

For more info about binding states, see Spring Cloud Stream: Binding visualization and control.

Metrics

This connector uses Spring Boot Metrics which leverages Micrometer to manage its metrics.

Connector Meters

In addition to the meters already provided by the Spring framework, this connector introduces the following custom meters:

Name	Type	Tags	Description	Notes

`solace.connector.process`	`Timer`	`type: channel`
		
		`name: <bindingName>`
		
		`result: (success|failure)`
		
		`exception: (none|exception simple class name)`
			Processing time	This meter is a rename of `spring.integration.send` whose `name` tag matches a binding name.

`solace.connector.error.process`	`Timer`	`type: channel`
		
		`name: <bindingNames>`
		
		`result: (success|failure)`
		
		`exception: (none|exception simple class name)`
			Error processing time	This meter is a rename of `spring.integration.send` whose `name` tag matches an input binding's error channel name (`<destination>.<group>.errors`).
				
				Meters might be merged under the same `name` tag (delimited by `|`) if multiple bindings have the same error channel name (i.e. bindings have matching `destination` and/or `group`). **Though a reminder that setting a binding's `group` is not supported.**

`solace.connector.message.size.payload`	`DistributionSummary` Base Units: `bytes`	`name: <bindingName>`	Message payload size	

Name	Type	Tags	Description	Notes
solace.connector.message.size.total	DistributionSummary Base Units: bytes	name: <bindingName>	Total message size	
solace.connector.publish.ack	Counter Base Units: acknowledgments	name: <bindingName> result: (success|failure) exception: (none|exception simple class name)	Publish acknowledgment count	

The solace.connector.process meter with result=failure is not a reliable measure of tracking the number of failed messages. It only tells you how many times a step processed a message, how long it took to process that message, and if that step completed successfully.

Instead, it's recommended to use a combination of solace.connector.error.process and solace.connector.publish.ack to track failed messages.

Add a Monitoring System

By default, this connector only includes JMX as its supported monitoring system.

To add additional monitoring systems, add the system's micrometer-registry-<system> jar and its dependency jars to the connector's classpath. The included systems can then be individually enabled/disabled by setting management.metrics.export.<system>.enabled=true in the application.yml.

Security

Securing Endpoints

Exposed Management Web Endpoints

By default, this connector only enables the `health` and `leaderelection` management endpoints. Where for the `health` endpoint, only the root status is returned by default (i.e. no health details).

To enable other management endpoints, see Spring Actuator Endpoints.

Authentication & Authorization

For this release, the connector only supports basic HTTP authentication.

By default, no users will be created unless the operator configures it in their config. Configuration parameters responsible for security are:

```
solace:
  connector:
    security:
      enabled: true
      users:
      - name: user1
        password: pass
      - name: admin1
        password: admin
        roles:
        - admin
```

In the above example, we have created 2 users:

- **user1**: Has access to perform GET (Read) requests.
- **admin1**: Has access to perform GET and POST (Read & Write) requests.

To fully disable security and permit anyone to access the connector's web endpoints, operators can configure the parameter `solace.connector.security.enabled` switched to `false`.

While these properties could be defined in an `application.yml` file, we recommend that you use environment variables to set secret values.

Here is an example of how to define users using environment variables:

```
# Create user with no role (i.e. read-only)
SOLACE_CONNECTOR_SECURITY_USERS_0_NAME=user1
SOLACE_CONNECTOR_SECURITY_USERS_0_PASSWORD=pass
```

```
# Create user with admin role
SOLACE_CONNECTOR_SECURITY_USERS_1_NAME=admin1
SOLACE_CONNECTOR_SECURITY_USERS_1_PASSWORD=admin
SOLACE_CONNECTOR_SECURITY_USERS_1_ROLES_0=admin
```

In the above example, we have created 2 users:

- **user1**: Has access to perform GET (Read) requests.
- **admin1**: Has access to perform GET and POST (Read & Write) requests.

`solace.connector.security.users` is a list. When users are defined in multiple sources (different `application.yml` files, environment variables, etc), then overriding works by replacing the entire list. Meaning that you must pick one place to defined your users and define all of them there. Whether it be in a **single** application properties file or all of them in the environment variables.

See Spring Boot - Merging Complex Types for more info.

CSRF Protection

Spring Boot enables CSRF protection by default on all management endpoints (see Spring Cross Site Request Forgery Protection). Though this connector disables CSRF protection for all POST requests on actuator endpoints so that users with write permissions (those with the `admin` role) can perform POST requests.

To fully disable CSRF protection, set the following config option:

```
solace.connector.security.csrf-enabled=false
```

TLS

TLS is disabled by default.

To configure TLS, see Spring Boot - Configure SSL and TLS Setup in Spring.

Consuming Object Messages

For the connector to process object messages, it needs access to the classes which define the object payloads.

Assuming that your payload classes are in their own project(s) and are packaged into their own jar(s), place these jar(s) and their dependencies (if any) onto the connector's classpath.

It is recommended that these jars only contain the relevant payload classes to prevent any oddities.

In the jar(s), your class files must be archived in the same directory/classpath as the application that publishes them.

e.g. If the source application is publishing a message with payload type, `MySerializablePayload`, defined under classpath `com.sample.payload`, then when packaging the payload jar for the connector, the `MySerializablePayload` class must still be accessible under the `com.sample.payload` classpath.

Typically, build tools such as Maven or Gradle will handle this when packaging jars.

Adding External Libraries

The connector jar uses the `loader.path` property as the recommended mechanism for adding external libraries to the connector's classpath.

See Spring Boot - PropertiesLauncher Features for more info.

To add libraries to the connector's container image, see the connector's container documentation.

Configuration

Providing Configuration

See Spring Boot: Externalized Configuration for info about how the connector will detect configuration properties.

Converting Canonical Spring Property Names to Environment Variables

See the Spring documentation for how to provide configuration options as environment variables.

Spring Profiles

If multiple config files will exist within the same config folder for use in different environments (dev, prod, etc), then use Spring profiles.

This will allow you to define different application property files under the same directory using the file name format, `application-{profile}.yml`.

eg:

- `application.yml`
  - Properties in non-specific files always applies. It's properties are overridden by those defined in profile-specific files.
- `application-dev.yml`
  - Defines properties specific to the `dev` environment.
- `application-prod.yml`
  - Defines properties specific to the `prod` environment.

Individual profiles can then be enabled by setting the `spring.profiles.active` property.

See Spring Boot: Profile-Specific Files for more info as well as an example.

Configure Locations to Find Spring Property Files

By default, the connector will detect any Spring property files as per Spring Boot's default locations.

- If you want to add additional locations, add `--spring.config.additional-location=file:<custom-config-dir>` (similar to the example command in Quick Start: Running the connector via command line).
- If you want to exclusively use the locations that you've defined and ignore Spring Boot's default locations, add `--spring.config.location=optional:classpath:/,optional:classpath:/config/,file:<custom-config-dir>`.

See Spring Boot documentation for more info.

If config files for multiple, different, connectors will exist within the same config folder for use in different environments (e.g. dev, prod, etc), then consider using Spring Boot Profiles instead of child directories to do this.

i.e.:

- Do this:
  - `config/application-prod.yml`
  - `config/application-dev.yml`
- Instead of this:
  - `config/prod/application.yml`
  - `config/dev/application.yml`

Child directories are intended to be used for merging configuration from multiple sources of config properties. For more information and an example of when you might want to use multiple child directories to compose your application's configuration, please see the Spring Boot documentation.

Spring Configuration Options

This connector packages a lot of libraries to customize functionality. Here are some references to get started:

- Spring Cloud Stream
- Spring Cloud Stream Binder for Solace PubSub+
- Spring Logging
- Spring Actuator Endpoints
- Spring Metrics

Connector Configuration Options

These configuration options are all prefixed by `solace.connector.`:

Config Option	Type	Valid Values	Default Value	Description
`management.leader-election.fail-over.max-attempts`	`int`	`> 0`	`3`	The maximum number of attempts to perform a fail-over.
`management.leader-election.fail-over.back-off-initial-interval`	`long`	`> 0`	`1000`	The initial interval (milliseconds) to back-off when retrying a fail-over.

Config Option	Type	Valid Values	Default Value	Description
`management.leader-election.fail-over.back-off-max-interval`	`long`	`> 0`	`10000`	The maximum interval (milliseconds) to back-off when retrying a fail-over.
`management.leader-election.fail-over.back-off-multiplier`	`double`	`>= 1.0`	`2.0`	The multiplier to apply to the back-off interval between each retry of a fail-over.
`management.leader-election.mode`	`enum`	`(standalone|active_active|active_standby)`	`standalone`	The connector's leader election mode. **standalone:** A single instance of a connector without any leader election capabilities. **active_active:** A participant in a cluster of connector instances where all instances are active. **active_standby:** A participant in a cluster of connector instances where only one instance is active (i.e. the leader), and the others are standby.
`management.queue`	`string`	`any`	`null`	The management queue name.
`management.session.*`		See Spring Boot Auto-Configuration for the Solace Java API		Defines the management session. This has the same interface as that used by `solace.java.*`. See Spring Boot Auto-Configuration for the Solace Java API for more info.
`security.enabled`	`boolean`	`(true|false)`	`true`	If `true`, security is enabled. Otherwise, anyone has access to the connector's endpoints.

Config Option	Type	Valid Values	Default Value	Description
security.csrf-enabled	boolean	(true|false)	true	If true, CSRF protection is enabled. Makes sense only if solace.connector.security.enabled is true.
security.users[<index>].name	string	any	null	The name of this user.
security.users[<index>].password	string	any	null	The password of this user.
security.users[<index>].roles	list<string>	admin	empty list (i.e. read-only)	The list of roles which this user has. Has read-only access if no roles are given.

Workflow Configuration Options

These configuration options are defined under the prefix, solace.connector.workflows.<workflow-id>. (if they support per-workflow config), and the default prefix, solace.connector.default.workflow. (if they support default workflow config).

Config Option	Applicable Scopes	Type	Valid Values	Default Value	Description
enabled	Per-Workflow	boolean	(true|false)	false	If true, the workflow is enabled.
transform-headers.expressions	Per-Workflow Default	Map<string,string>	**Key:** A header name. **Value:** A SpEL string which accepts headers as parameters.	empty map	A mapping of header names to header value SpEL expressions. The SpEL context contains the headers parameter which can be used to read the input message's headers.

include::../../../snippets/attributes/common.adoc

Database Source Configuration Options

These configuration options are all prefixed by solace-persistence.:

Config Option	Type	Valid Values	Default Value	Description
datasource	embedded config	see Spring Datasource Configuration		The spring datasource configuration to read data
jpa	embedded config	see Spring JPA Configuration		The spring jpa configuration
source.strategy	string	readingIndicator,timestamp,sequential	readingIndicator	pulling strategy. readingIndicator: pulling data on condition of column specified by DB_READ_INDICATOR_COLUMN , reading DEFULT value and setting to PROCESSED after pulling; timestamp: pulling data on condition of column specified by DB_READ_RECORD_TIMESTAMP_INDICATOR, ranged by timestampPolling; sequential: pulling data sequentially by values of column specified by DB_READ_RECORD_SEQUENTIAL_ORDER;
source.dbQueryMode	string	jpa or sql(not fully supported)		query using jpa or sql
source.sendBatchSize	int	send in batch		currently only support sending one by one
source.sendPoolSize	int	max 255	100	the thread pool size used for sending messages
source.queryMax	int	query records a time	1000	query from table for the records number a time
source.triggerInterval	long	long	1000	interval of each db query, in millisecond
source.enableRetry	string	true or false	true	if true, will retry to do db query
source.retryInterval	int	int	5	interval of db query on error occurred, in second

Config Option	Type	Valid Values	Default Value	Description
source.retryCount	int	int	3	retry times to db query on error occurred, in second
source.enableBufferedJpaBatch	string	true or false	true	if true, connector will write back indicator values in batch mode
source.bufferWaitSeconds	int	int	5	max time to wait for the batch write, in second
source.enableDatabaseTimestamp	string	true or false	true	if true, will include the reading DB record time in the message header.
source.enableConnectorTimestamp	string	true or false	true	if true, will include the connector sending message time in the message header.
source.enableBenchmarkInfo	string	true or false	false	if true, print the performance statistics.
source.dbQueryMode	string	jpa or sql	jpa	the internal implementation of db query
source.skipMismatchError	boolean		false	if true, skip update db rows mismatch errors
source.indicatorFlag	boolean		true	ture for indicatorFlag rewrite , false for not
source.timestampPolling	long		6000	timestamp range of each pulling, in milliseconds
source.timeWindow	long		0	extra timestamp range of each pulling, in milliseconds

Database Source Configuration in connector.properties

Config Option	Type	Valid Values	Default Value	Description
<table_name>:entity mapping	string	<table_name>:entity mapping		Fill in table name mapped to a entity
table_name.BINDING	string	output-0	output-0	use this default value

Config Option	Type	Valid Values	Default Value	Description
table_name.TOPIC	string	topic name with multiple levels, using /		data topic name, send the event of record to this topic
table_name.LVQ_TOPIC	string	topic name with multiple levels, using /		lvq topic name, send checkpoint data to this topic
table_name.DB_TABLE	string	parent table		parent table name
table_name.DB_PARENT_KEY	string	parent table key		key column property in the parent table, has relationship with child table key.
table_name.DB_PARENT_COLLECTION	string	map object		map object in the parent entity
table_name.DB_CHILD_TABLE	string	child table name		child table name
table_name.DB_CHILD_KEY	string	child table key		key column property in the child table, has relationship with parent table key
table_name.DB_PARENT_CHILD_MODE	string	one2many or many2one	one2many	the parent-clild table relationship mode
table_name.DB_READ_INDICATOR_COLUMN	string	source table indicator column		source table read indicator column
table_name.DB_TRACKING_ID	string	tracking id value		specify the tracking ID column property, disabled when value not presented
table_name.DB_READ_TIME	string	connector will update this column when reading this record		specify the read time column property, update this column with the reading record time by connector. disabled when value not presented
table_name.DB_READ_INDICATOR_COLUMN_UPDATE_VALUE_INPROGRESS	string	indicator column will change to this value		after records read by connector and in the process

Config Option	Type	Valid Values	Default Value	Description
table_name.DB_READ_INDICATOR_COLUMN_UPDATE_VALUE_PROCESSED	string	indicator column will change to this value		after records read by connector, sent to solace as event and got the ack.
table_name.DB_READ_INDICATOR_COLUMN_DEFAULT_VALUE	string	default value		initial value and ready to read by connector.
table_name.DB_READ_INDICATOR_COLUMN_FAILED_VALUE	string	set to a failed value		error or failed during processing.
table_name.DB_READ_RECORD_SEQUENTIAL_INDICATOR	string	table column property		table records reading sequencial column property
table_name.DB_READ_RECORD_SEQUENTIAL_ORDER	string	asc or desc	asc	read table records
table_name.DB_READ_INDICATOR_BRIDGE_FAILED_VALUE	boolean	true or false	false	if true, it will read the failed ones after starting the connector, if failed, it will ignore the failed one.
table_name.MESSAGE_HEADER_<custom_header>	string	header name		it will have custom_header in the message property, its value is from the record of the column property specified
table_name.ENABLE_<property_name>	boolean	true or false	none	if false , target property must be configured as empty value; if true, target property must be configured and not empty

include::../../../snippets/attributes/common.adoc

Database Sink Configuration Options

These configuration options are all prefixed by `solace-persistence.`:

Config Option	Type	Valid Values	Default Value	Description
datasource	embedded config	see Spring Datasource Configuration		The spring datasource configuration to write data

Config Option	Type	Valid Values	Default Value	Description
jpa	embedded config	see Spring JPA Configuration		The spring jpa configuration
sink.jpaBatchMode	string	true or false	true	true to enable batch insert to the DB table
sink.jpaBatchSize	int	25,100…	25	Set the batch size to DB when jpaBatchMode is true
sink.jpaMaxRetry	int	2,3,4…	3	Retry times to insert to DB table times.
sink.jpaRetryWaitMilliseconds	int	3000	empty	Interval between retry, milliseconds
sink.redirectOnFailEnable	string	true or false	true	when true, if the topic name in the queue is not mapped to any table, it will redirect this message to another new topic with a redirectPrefix prepend to the original topic name
sink.redirectPrefix	string	<string>/	prefix/, could be any string + /	prepand to the original topic name
sink.payloadFormat	string	xml or json	xml	payload format sent to Solace queue

Database Sink Configuration in connector.properties

Config Option	Type	Valid Values	Default Value	Description
table_name:enity mapping	string	<table_name>:<entity class reference>	empty	ex.customer:entity.sink.database.com.solacecoe.connectors.Customer
topic:table_key mapping	string	<topic>:<table_name>	empty	set topic name to table name mapping, the event of contain this topic name will be insert into the table. wildcard >,* can be used for filter the topic.
DB_CONNECTOR_TIMESTAMP_FORMAT	string	timestamp format	yyyy/dd/MM.HH:mm:ss	Set the timestamp format pattern

Config Option	Type	Valid Values	Default Value	Description
DB_CONNECTOR_BLOB_ENCODING	string	BASE64 or HEX	BASE64	Blob Encoding format in the message payload
DB_MSG_HEADER_TIMESTAMP_FORMAT	string	timestamp format	yyyy/dd/MM.HH:mm:ss	Set the message header timestamp format pattern
MESSAGE_HEADER_<header property>=<table name>.column property	string	<table_name>.column property name	empty	For example: MESSAGE_HEADER_TRACK_ID=passenger.day, if message header has a property "TRACK_ID", its value will inserted to table "passenger", column "day"

More information

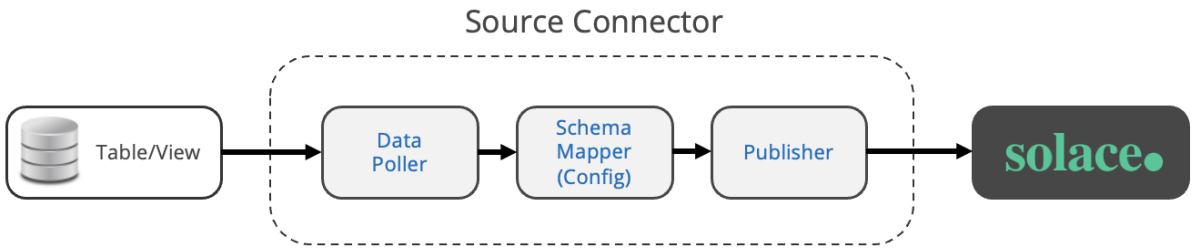
Solace Pubsub Connector For Database Source

1 Introduction

The **Solace Pubsub Connector For DB Source** has been built to make DB transactions participated in event mesh by using the Connector for DB Source

Solace Pubsub Connector For DB Source helps to read the database records and make it available as an event in the Solace Event Mesh for enterprise applications to consume.

Note: Pulls the records which are marked as unread.



1.1 Purpose

To guide the technical teams to setup the **Solace Pubsub Connector For DB Source**.

1.2. Scope

Covering the step-by-step guide to setup the **Solace Pubsub Connector For DB Source**.

1. Generate JPA Entities using the tool
2. Configure DB Souce Connector
    1. Connection Configuration (Database & Solace)
    2. Connector & Entity Configuration
    3. Schema Mapping (Table Structure to Solace Event Payload)

`Note: This document currently captures the deployment in Java environment.`

1.3 DB Connector Features

S.No	Feature
1	The source DB object from where Adapter polls remains as-is (as EMS). No change expected at application DB end (Includes Tables, Views & Child Table)
2	Adapter should pick records in sequence only (Based on Sequence Number and timestamp field)
3	The Adapter should be able to update each record with appropriate flag (in existing adb_l_delivery_status field) N - New : set by trigger P - Pending: under process (picked up by adapter, not yet published to messaging layer) C - Completed: Processing completed (published successfully to messaging layer) F - Failed: Fail to publish record
4	The adapter should have the capability to read in batch and update flag in batch to avoid chattiness and ensure better performance
5	Oracle DB should be supported (12g onwards (19C and ADG and RAC)
6	Design should be future proof for oracle version upgrades. Should support Thick and Thin DB driver
7	Should support ADG URL
8	Should have option to retry before marking record as 'F' The query to poll should be configurable.
9	Schema mapping feature should be Forward compatible.
10	Any new addition to DB schema should be ignored by Adapter unless Field name and mapping is provided.
11	Option to specify the data type of each field in the mapping.
12	Support for CLOB and BLOB
13	Date format to be as per the TIBCO specs - Should be configurable. default is yyyy-MM-dd’T’HH:mm:ss.SSS
14	Logging & Syslog

S.No	Feature
15	High Available & Fault Tolerance
16	Event Headers: In Solace Message Header, for each message published, add Tracking Id coming from source table Time at which picked from DB Time of publish to Solace

2 DB Connector Configuration

2.1. Pre-requisites

- JDK 17
- Maven 3.8+
- Solace
- Database for Oracle or MySQL

2.2. Install Solace Framework and Transform Engine

Go to the Framework jar and pom files folder and run below command to install into your local maven repo:

```
mvn install:install-file
-Dfile=solace-transformation-engine-1.0.10-SNAPSHOT.jar
-DgroupId=com.solace.connector.core
-DartifactId=solace-transformation-engine  -Dversion=1.0.10-SNAPSHOT
-DpomFile=solace-transformation-engine-1.0.10-SNAPSHOT.pom
```

2.3. Solace Pubsub Connector For DB Source

Download the Solace DB connector (spring-cloud-stream-binder-db and pubsubplus-connector-database) and unzip to the location/machine where the connector is getting installed

Go to spring-cloud-stream-binder-db folder and run:

```
mvn -DskipTests=true install
```

Go to pubsubplus-connector-database folder and run:

```
mvn -DskipTests=true clean package
```

2.4 DB Configuration

Cofiguration File: `projConfig/application.yml`

```
solace-persistence:
  datasource:
    driver-class-name: oracle.jdbc.OracleDriver
    url: jdbc:oracle:thin:@//192.168.3.178:1524/ORCLCDB
    username: c##test
    password: test
    hikari:
      initializationFailTimeout: -1
      connection-timeout: 5000
  jpa:
    hibernate:
      ddl-auto: none
      show-sql: false
    properties:
      hibernate:
        jdbc: batch_size: 100
        order_updates: true
        generate_statistics: false
      dialect: org.hibernate.dialect.Oracle10gDialect
      database: oracle
  source:
    dbQueryMode: jpa # jpa or sql(not fully supported)
    sendBatchSize: 1 #records per message
    sendPoolSize: 100 # send thread pool size, defalut 100 max 255
    queryMax: 2 #limited records per DB query
    maxDbRetry: 3
    triggerInterval: 2000
    retryInterval: 10 # seconds
    enableRetry: true #enable retry to send the message to Solace.
    retryCount: 3
    enableBufferedJpaBatch: true # only available when sendBatchSize = 1
    bufferWaitSeconds: 5
    enableTrackingId: true #update tracking Id to table
    enableDatabaseTimestamp: true
    enableConnectorTimestamp: true
    enableBenchmarkInfo: true
    skipMismatchError: false
    indicatorFlag: true
```

- Fill in all the required details of `solace-persistence.datasource` to establish a connection with Database.
- `solace-persistence.datasource` section, `batch_size`\: Given batch size will allow connector to update the status back to Database in a batch mode
- `solace-persistence.datasource.hikari` section, Hikari Settings: Will enable connector to perform retry/reconnect until it is timeout.
- `solace-persistence.source` section, Additional configurations to pull the data from Database.

2.5 Solace Configuration

Configuration File: `projConfig/application.yml`

```
spring:
  cloud:
    stream:
      bindings:
        input-0:
          destination: souceBinderLVQ #LVQ queue name
          binder: solace-db #Solacedatabase binder
          group: no
        output-0:
          destination: dxb/passenger #topic to send data
          binder: solace
          content-type: application/x-java-serialized-object
```

- `spring.cloud.stream.bindings.input-0` section, Fill in all the required details to establish a connection with Solace
- `spring.cloud.stream.bindings.output-0` section, Set contentType to application/x-java-serialized-object to send plain text message, the default is bytes.

Configure connection to Solace:

```
  java:
    host: tcp://192.144.217.87:55555
    msgVpn: default
    clientUsername: admin
    clientPassword: admin
    connectRetries: -1
    reconnectRetries: -1
    apiProperties:
      pub_ack_window_size: 50
      pub_ack_time: 200
```

Configure the management part for the HA:

```
solace:
  connector:
    workflows: # Workflow configuration
      0:
        enabled: true  # If true, the workflow is enabled.
        acknowledgment:
          publish-async: true
        transformation:
          mappingFile: ""
          mode: customentitytoxml
    #      1:
```

```
    #        enabled: false  # If true, the workflow is enabled.
    #        acknowledgment:
    #          publish-async: true
    management:
      leader-election:
        mode: active_standby  # The connector's leader election mode. (values:
standalone, active_active, active_standby)
        fail-over:
          max-attempts: 3  # The maximum number of attempts to perform a fail-over.
          back-off-initial-interval: 1000  # The initial interval (milliseconds) to
back-off when retrying a fail-over.
          back-off-max-interval: 10000  # The maximum interval (milliseconds) to back-
off when retrying a fail-over.
          back-off-multiplier: 2.0  # The multiplier to apply to the back-off interval
between each retry of a fail-over.
      queue: management-queue-push  # The management queue name.
      session: # The management session. This has the same interface as that used by
`solace.java.*`. For more info: https://github.com/SolaceProducts/solace-spring-
boot/tree/master/solace-spring-boot-starters/solace-java-spring-boot-starter#updating-
your-application-properties
        host: tcp://192.144.217.87:55555
        msgVpn: default
        client-username: admin
        client-password: admin
```

2.6. Config file and mapper file

Configuration File: `connector_config.properties\`:

```
#table entity mapping
source_passenger=entity.source.database.com.solacecoe.connectors.SourcePassenger
passAddress=entity.source.database.com.solacecoe.connectors.PassAddress

DB_TABLE=source_passenger
#--------------------source_passenger-------------
#<DB_TABLE>.ENABLE_<property>=true or false, if false , target property must be
configured as empty value; if true, target property must be configured and not empty
source_passenger.ENABLE_DB_READ_TIME=true
source_passenger.ENABLE_ DB_TRACKING_ID=false
#use default output-0
source_passenger.BINDING=output-0
#data topic name, send the event of record to this topic
source_passenger.TOPIC=dxb/source_passenger
#lvq topic name, send checkpoint data to this topic
source_passenger.LVQ_TOPIC=lvq/source_passenger
#parent table name
source_passenger.DB_TABLE=source_passenger
#DB_PARENT_CHILD_MODE: many2one or one2many, default is one2many
source_passenger.DB_PARENT_CHILD_MODE=many2one
#key column property in the parent table, have relationship with child table key.
```

```
source_passenger.PARENT_CHILD_RELATION_KEY=passengerId:passId,
nationality:zip,<parentKey>:<childKey>
#map in the main entity.
source_passenger.DB_PARENT_COLLECTION=addresses
#child table name
source_passenger.DB_CHILD_TABLE=passAddress
#source table read indicator column
source_passenger.DB_READ_INDICATOR_COLUMN=flag
#specify the tracking ID column property
source_passenger.DB_TRACKING_ID=dbtrackid
#specify the read time column property, update this column with the reading record
time by connector
source_passenger.DB_READ_TIME=readtime
# status of the indicator column value.
source_passenger.DB_READ_INDICATOR_COLUMN_UPDATE_VALUE_INPROGRESS=p
source_passenger.DB_READ_INDICATOR_COLUMN_UPDATE_VALUE_PROCESSED=c
source_passenger.DB_READ_INDICATOR_COLUMN_DEFAULT_VALUE=n
source_passenger.DB_READ_INDICATOR_COLUMN_FAILED_VALUE=f
# sequeial column property
source_passenger.DB_READ_RECORD_SEQUENTIAL_INDICATOR=id.passengerId
# order by, asc or desc
source_passenger.DB_READ_RECORD_SEQUENTIAL_ORDER=asc
#if true, it will read the failed ones after starting the connector, if failed, it
will ignore the failed one.
source_passenger.DB_READ_INDICATOR_BRIDGE_FAILED_VALUE=true
##message heasers
source_passenger.ENABLE_MESSAGE_HEADER_TRACK_ID=true
source_passenger.MESSAGE_HEADER_TRACK_ID=dbtrackid
#MESSAGE_HEADER_CREATE_TIME=createTime
```

Create mapper files for source_<table_name>_trgt_schema_mapper.yml for each table, ex source_passenger_trgt_schema_mapper.yml</table_name>

```
solace-connector-mapper:
  mapper-metadata:
    escape-special-characters: true
    schema-type:
    encrypted:
  payload-mapper:
    mapper-name0:
      source-xpath: source_passenger/passengerId
      source-datatype:
      target-xpath: passenger/id
      target-datatype:
    mapper-name1:
      source-xpath: source_passenger/name
      source-datatype:
      target-xpath: passenger/pname
      target-datatype:
    mapper-name2:
```

```
      source-xpath: source_passenger/birthday
      source-datatype:
      target-xpath: passenger/birthday
      target-datatype:
    mapper-name3:
      source-xpath: source_passenger/passAddress/city
      source-datatype:
      target-xpath: passenger/addresses/passAddress/city
      target-datatype:
```

2.7 Connector Logging

Logback has been used to enable the logging mechanism.

Configuration File: `resourceslogback.xml`

You can change your log default folder in logback.xml:

```
<property name="LOG_HOME" value="log/pull-connector/" />
```

You can change your log level as showed below:

```
<!--DEBUG,INFO,WARN,ERROR,FATAL-->

<root level="INFO">
  <appender-ref ref="STDOUT" />
  <appender-ref ref="FILE" />
  <appender-ref ref="RSYSLOG" />
</root>
```

2.8 Admin client configuration

Please configure the admin sever as follows in application.yml

```
spring:
  boot:
    admin:
      client:
        enabled: true
        url: http://localhost:8082       #configure your admin server url
        auto-registration: true
  application:
    name: Instance006                    #configure your application name
```

3 Generate Entity tool.

Hibernate API's has been used to generate the JPA entities and load into the connector code. This will reduce the manual efforts to write the entity file for all the tables.

Note: We are also finding a way to automate the entire entity generation process to reduce the manual interventions.

Download the generateEntity project and edit the below show file to generate the entities on the connected schema.

`pom.xml` configuration

```
<configuration>
    <!--
<templatePath>${project.basedir}/src/main/resources/jpagen/tempalate/</templatePath>--
>
    <!-- Defaults: -->
    <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
    <!--hibernate configuration file-->
    <propertyFile>src/main/resources/hibernate.properties</propertyFile>
    <!--DB and Entity data type mapping configuration file-->
    <revengFile>src/main/resources/reveng.xml</revengFile>
    <!--entity operation info -->
    <revengStrategy></revengStrategy>
    <packageName>com.solace.connector.db.pull.entity</packageName>
    <detectManyToMany>true</detectManyToMany>
    <detectOneToOne>true</detectOneToOne>
    <detectOptimisticLock>true</detectOptimisticLock>
    <createCollectionForForeignKey>true</createCollectionForForeignKey>
    <createManyToOneForForeignKey>true</createManyToOneForForeignKey>
    <!-- true JPA anotation -->
    <ejb3>true</ejb3>
    <jdk5>true</jdk5>
</configuration>
```

DB configuration The file `hibernate.properties` location is configured in pom.xml file:

```
default\_schema=TEST
hibernate.connection.driver\_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.username=system
hibernate.connection.password=oracle
hibernate.connection.url=jdbc:oracle:thin:@localhost:47161/xe
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

Data type mapping configuration The file `reveng.xml` location is configured in pom.xml file:

```
<sql-type jdbc-type="DATE" hibernate-type="java.util.Date"/>
    <sql-type jdbc-type="TIMESTAMP" hibernate-type="java.util.Date"/>
```

```
    <!--<sql-type jdbc-type="NUMERIC" hibernate-type="java.lang.Double"/>-->
    <sql-type jdbc-type="DECIMAL" hibernate-type="java.lang.Double" />
    <sql-type jdbc-type="CHAR" hibernate-type="java.lang.String" />
    <sql-type jdbc-type="NUMERIC" length="10" precision="3"  hibernate-
type="java.math.BigDecimal" />
    <sql-type jdbc-type="NUMERIC" length="20" precision="5"  hibernate-
type="java.math.BigDecimal" />
    <sql-type jdbc-type="NUMERIC" length="37" precision="4"  hibernate-
type="java.math.BigDecimal" />
    <sql-type jdbc-type="FLOAT"   hibernate-type="java.math.BigDecimal" />
    <sql-type jdbc-type="NUMERIC" length="38" precision="0"  hibernate-
type="java.math.BigDecimal" />
    <sql-type jdbc-type="NUMERIC"  hibernate-type="java.math.BigDecimal" />
    <sql-type jdbc-type="BLOB"   hibernate-type="java.lang.Byte[]" />
    <sql-type jdbc-type="CLOB"   hibernate-type="java.lang.String" />
```

Note:

- match-name could be a single table name, or it can be a wildcard string (.*) to generate the entity.
- Some of the minute modifications needed to fit to DB Connector entity requirements
- Also, no modification to single table entity, this is only for parent/child table.
- Add tablename to generate a single table

```
<table-filter match-schema=".*" match-name="tablename" />
```

Parent table ex:

```
// Below are generated by tool, should be removed:
@OneToMany(fetch=FetchType.LAZY, mappedBy="sourcePassenger")
private Set<PassAddress> passAddresses = new HashSet<PassAddress>(0);
Set<PassAddress> passAddresses
Set<PassAddress> getPassAddresses()
Set<PassAddress> passAddresses

Replace all the Set<> to be List<Map> in the entity:
private List<Map> addresses;
@JsonIgnore
@Transient
public List<Map> getAddresses() {
    return addresses;}
public void setAddresses(List<Map> addresses) {
    this.addresses = addresses;}
```

The `addresses` is the `DB_PARENT_COLLECTION` `of` `connector_config.properties` .

If there is no PK for parent table. 2 entities are generated for parent table, `entity` and `entityId`.

Move all the properties/methods from entityId to entity.

In the entity, remove `@AttributeOverrides({})`.

For example: Parent table entity: `SourcePassenger` Parent table entityId: `SourcePassengerId`

Move all the property in SourcePassengerId to SourcePassenger, and remove `@AttributeOverrides({})` in SourcePassenger and add below:

Add `@Id` on the unique column of `SourcePassenger`\:

```
@Id
@Column(name="PASSENGER_ID", nullable=false, precision=10, scale=0)
```

Child table ex:

If there is no PK for child table. 2 entities are generated for child table, `entity` and `entityId`. Move all the properties/methods from entityId to entity .

In the entity, remove `@AttributeOverrides({})`.

For example: Child table entity: `PassAddress` Child table entityId: `PassAddressId` Move all the properties in PassAddressId to PassAddress. Add @Id on the unique column of PassAddressId:

```
@Id
@Column(name="ID", nullable=false, precision=10, scale=0)
```

Create VIEW entity:

When it is a VIEW object in DB, please treat it as a single table, just point your table name to your view name.

For example, if you have a view `Customer`. 2 entities will be generated, `Customer` and `CustomerId`, please move all properties from CustomerId to Customer. In the entity, remove `@AttributeOverrides({})`. Add `@Id` on the unique column of `Customer`\:

```
@Id
@Column(name="CUSTOMER_ID", nullable=false, precision=10, scale=0)
```

Command to run:

```
mvn hibernate-tools:hbm2java
```

Generate the entity jar:

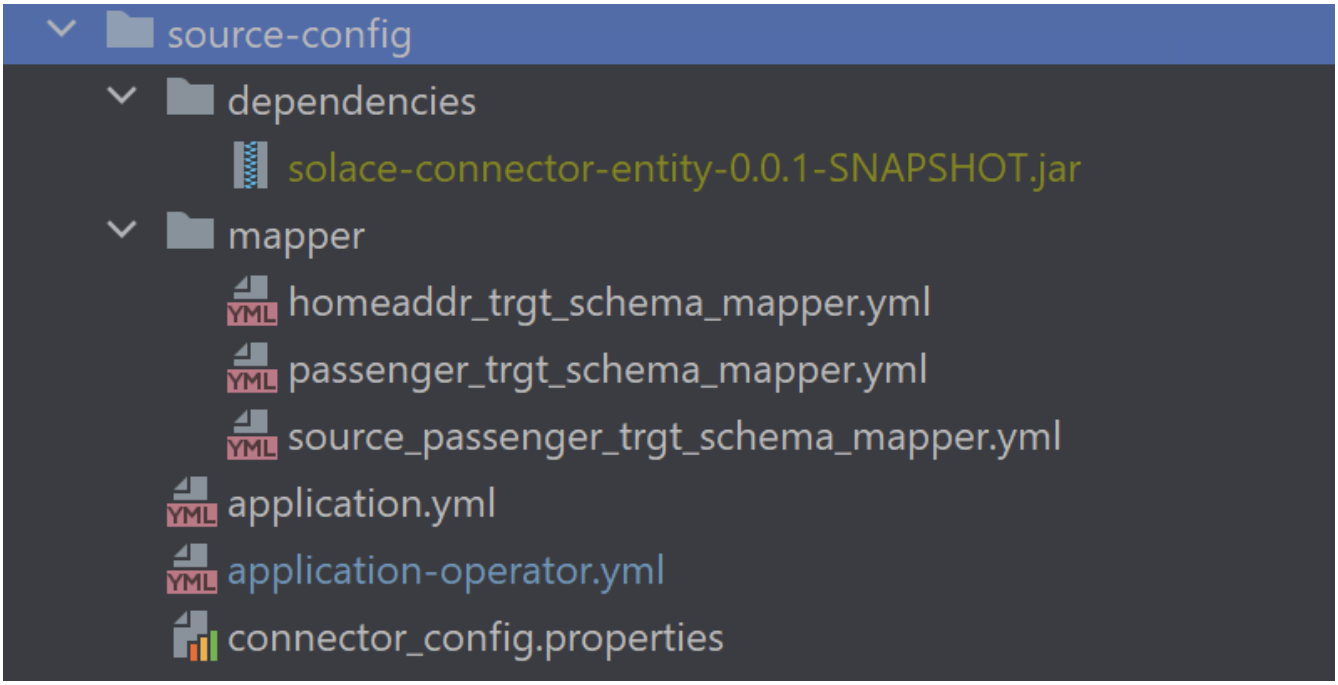
Put the entities to your entity project, manually add entity repo and generate the entity jar, put the entity jar to \\config\\dependencies folder.

4 Run Connector

4.1 Command line to run the Solace Pubsub Connector For DB source

Go the project

```
mvn clean package
```



Go to project folder and run as standalone mode:

```
java -cp ./target/pubsubplus-connector-database-1.2.0-SNAPSHOT.jar
-D"loader.path=./source-config/dependencies,./source-config"
org.springframework.boot.loader.PropertiesLauncher
--spring.config.additional-location="./source-config/application-operator.yml"
-server.port=8081
```

4.2 Command line to run the Solace Pubsub Connectors as HA

Please specify server.port to another value that is not used.

```
java -cp ./target/pubsubplus-connector-database-1.2.0-SNAPSHOT.jar
-D"loader.path=./source-config/dependencies,./source-config"
org.springframework.boot.loader.PropertiesLauncher
--spring.config.additional-location="./source-config/application-operator.yml"
-server.port=8082
```

Please refer to 1.4.2 Solace Configuration> Configure the management part for the HA.

4.3 Deploy as docker container and run

```
docker run --name pubsubplus-connector-database -v
D:\\GitHub\\docker\_config:/config pubsubplus-connector-database:v1
docker\_config refers to source-config and sink-config
```

5 Connector Monitoring

Connector monitoring is a Spring Boot Admin server which helps to visualize the connectors deployed in the organization

5.1 Prerequisites

- Maven 3
- JDK 17

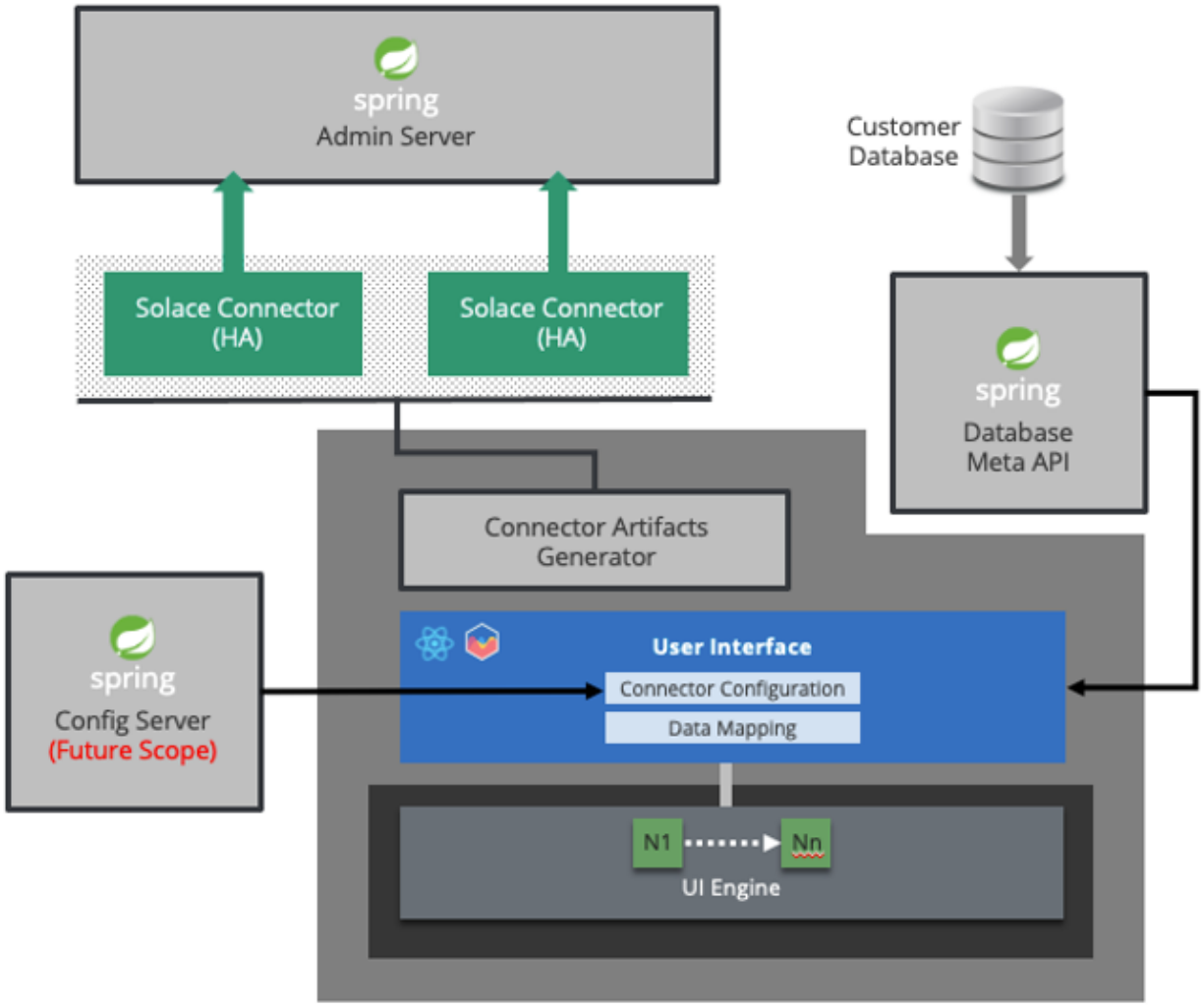
5.2 Infrastructure Requirement

Component Name	Cores	RAM (GB)	Runtime
Spring Boot Admin	2	4	Java

6 Connector Management

- Connector management is a UI based tool to generate the code binary based on the configuration provided in the tool.
- All the connector configurations will be stored in the database attached to the CM
- Will allow to edit the configuration for the available connectors in the CM database.

6.1 Architecture



6.2 Pre-Requisites

S.No	Runtime	Version	Description
1	NodeJS	V12	Used for CM runtime
2	NPM	V6	Used to manage node packages required by CM
3	PM2	5.2	Used to manage NodeJS process for scalabilty
4	Java	17	Used for Database Meta API runtime
5	Oracle Instant Client	21	Required for NodeJS library to initiate connection with Oracle
6	Maven	3	Used to Manage dependencies required by Meta API

6.3 Infrastructure Requirement

Solace Connector Management will be deployed in a single VM

Component Name	Cores	RAM (GB)	Runtime
Connector Management (UI+Services)	4	6	NodeJS
DB Meta API (Per Schema)	1	2	Java

6.4 Connector Management Installation Guide

CM Deployment kit contains:

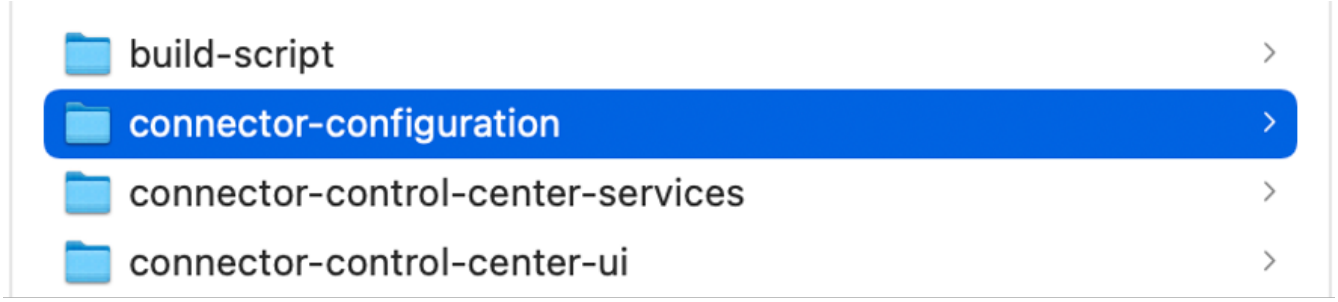
- Schema script (Oracle)
- This script includes all the required database commands to create tables required for Connector Management
- Spring Admin Server Binary
- This binary spins ups the Spring Admin server after providing required configurations
- CM NodeJS Binary
- User interface component which can be used to configure/manage connectors.

Database preparation

- Create Schema: `connector_control_schema`
- Create the tables: Execute the Schema Script provided in the kit to create the required tables.

Deploy CM:

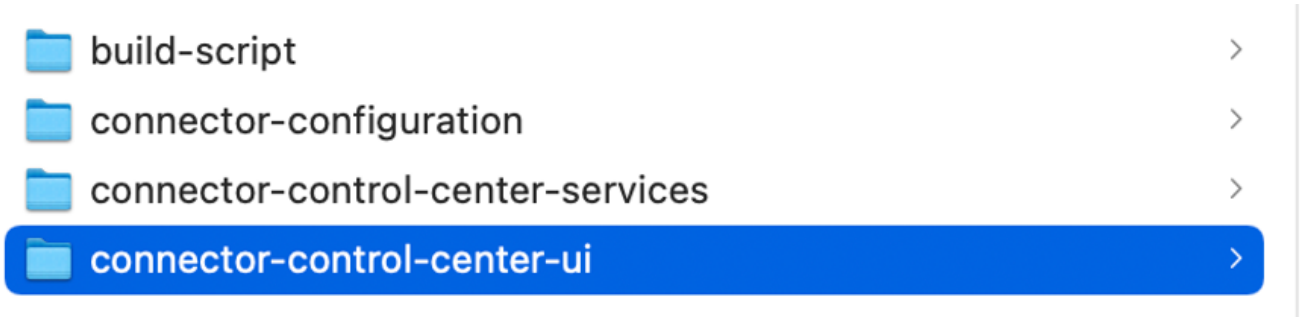
- Unzip the file <file-name> to the desired location</file-name>
- Edit the ecosystem.config.js file located in `connector_configuration` folder. This configuration is used by PM2



```
env_dev: {
  "NODE_ENV": "dev",
  "DATABASE_HOST": "",
  "DATABASE_PORT": 5432,
  "DATABASE_USER": "",
  "DATABASE_PASSWORD": "",
  "DATABASE_NAME": "",
  "CONTROL_CENTER_UI_DOMAIN": "",
  "CONTROL_CENTER_BACKEND_DOMAIN": "",
  "SECRET_KEY": "",
  "REFRESH_TOKEN_KEY": "",
  "API_PORT": 3001,
  "SOLACE_VPN_NAME": '',
  "SOLACE_HOST": '',
  "SOLACE_AUTH_TOPIC": "",
  "SOLACE_ACC_DATA_PROTOCOL": "",
  "SOLACE_ACC_REST_PORT": 9443,
  "SOLACE_CLIENT_USER": "",
  "SOLACE_CLIENT_PWD": "",
  "CCC_GENERATE_ENTITY_TEMPLATE_PATH": 'generateEntity.zip',
  "CCC_SOLACE_DB_PULL_CONNECTOR_TEMPLATE_PATH": 'solace-db-pull-connector.zip',
  "CCC_DOWNLOAD_GENERATE_ENTITY_PATH" : "deploy-connectors"
},
```

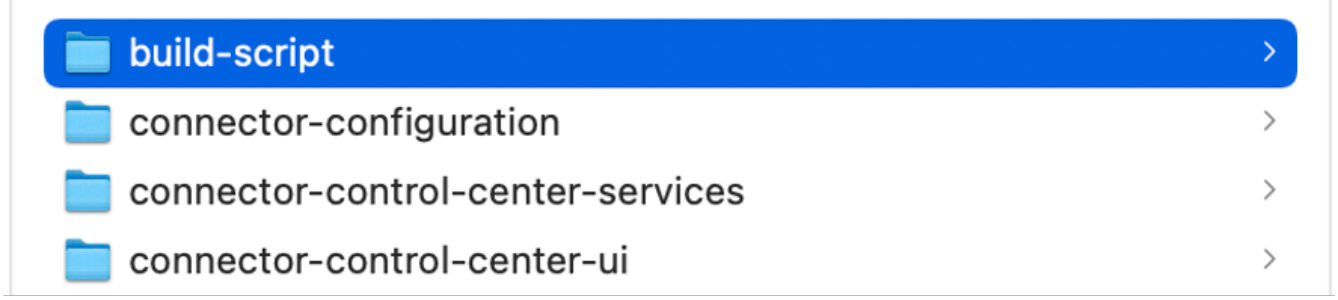
The same configuration can be repeated for different environments like dev, qa, prod.

Edit the `.env` file in the UI folder `connector-control-center-ui`




```
REACT_APP_API_SERVICES_URL = 'http://<host-name>:3001'
REACT_APP_DOMAIN = 'http://<host-name>:3000'
#### CCC META API
REACT_APP_CCC_META_API_URL = 'http://<host-name>:8082'
```

Run the script located in `build-script` folder in the kit to create the build.



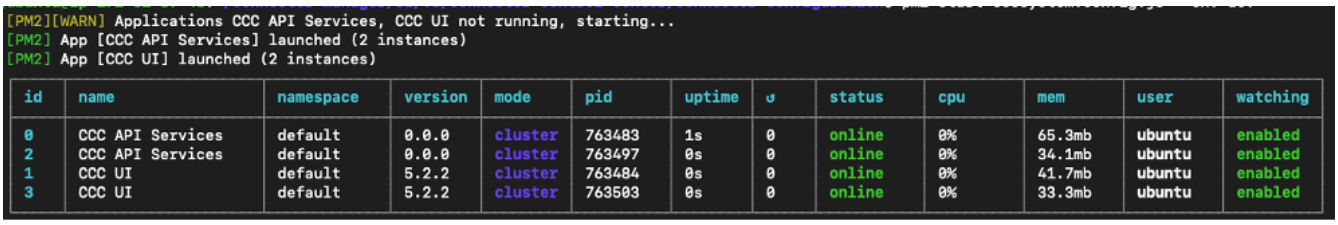
Command to execute the script

```
cc_deploy.sh
```

Once successfully executed the script navigate to connector-configuration folder and execute the below command

```
pm2 start ecosystem.config.js env <dev,qa,prod>
```

To validate, run `pm2 ls` and you should see the following services up and running.



6.5. Connector Generation Guide

Generating the connector binary is a three-step process

1. Add schema to CM
2. Generate entity (JPA Notation) for the schema
3. Generate connector configurations as per the selected schema tables to bridge the data to Solace as events

6.6. Add Schema

6.7. Generate Entity

6.8. Generate Connector Configurations

7 Connector Config Parameters

7.1. Connector parameters

Config Type	Config Location	Parameter (Hierarchy)	Description
Generic-DB	application.yml	solace-persistence:datasource:driver-class-name	DB Connection Details
solace-persistence:datasource:url			
solace-persistence:datasource:username			
solace-persistence:datasource:password			
solace-persistence:datasource:hikari:minimum-idle	Config the connection pool for DB		
solace-persistence:datasource:hikari:idle-timeout			
solace-persistence:datasource:hikari:maximum-pool-size			
solace-persistence:datasource:hikari:max-lifetime			
solace-persistence:datasource:hikari:connection-timeout			
solace-persistence:jpa:hibernate:ddl-auto	hibernate property, recommended to use none in production		
solace-persistence:jpa:show-sql	True, will print the queries in the logs		
solace-persistence:jpa:database	Type of the target database ex: mysql,oracle		

Config Type	Config Location	Parameter (Hierarchy)	Description
solace-persistence:jpa:properties:jdbc:batch_size	The size of jpa batch opration		
spring:cloud:function:stream:poller:fixed-delay	Defines the interval frequency of the data polling from Database		
spring:cloud:function:stream:bindings:dataCollectting-out-0:destination	Allowed to configure both static and dynamic topics Some of the parameterized notations will add data to topic during the runtime. {COL:column name}: adds the value for defined column name into the to the topic {DB_TABLE}: adds a table name to the topics based on connector_config.properties file Ex: DXB/{COL:passengerId}/{COL:name}/SGP/{DB_Table}		
logging	logback.xml	<root level="INFO"> <appender-ref ref="STDOUT" /> <appender-ref ref="FILE" /> <appender-ref ref="RSYSLOG" /> </root>	used logback to config logging, it has different levels: DEBUG,INFO,WARN,ERROR,FATAL
Source property	application.yml	solace-persistence:source:jpaBatchMode	Allow to merge multiple records into a single event
solace-persistence:source:queryMax	Set the max records for each query read from database		
solace-persistence:source:maxDbRetry	it will do retry to connect the db		

Config Type	Config Location	Parameter (Hierarchy)	Description
solace-persistence:source:enableRetry	If true, it will retry query to db		
solace-persistence:source:retryInterval	Messages resent to Solace in the defined interval		
solace-persistence:source:retryCount	Max retry count and mark the record as a failure attempt		
solace-persistence:source:enableBufferedJpaBatch	if true, connector will write back indicator values in batch mode		
solace-persistence:source:bufferWaitSeconds	timeout for getting ack from Solace		
solace-persistence:source:enableRetry	True, to enable resend the message to Solace		
solace-persistence:source:retryCount	When enableRetry is true, retry numbers to the Solace.		
solace-persistence:source:enableBufferedJpaBatch	True to enable batch update to the DB		
solace-persistence:source:bufferWaitSeconds	Interval between batch update to the DB		
solace-persistence:source:enableTrackingId solace-persistence:source:enableDatabaseTimestamp solace-persistence:source:enableConnectorTimestamp	True, to enable message header contain those info.		
solace-persistence:source:enableBenchmarkInfo	True, to print performance detail log		
Admin Client	application.yml	sping.boot.admin.client.enable:true	True to enable.

Config Type	Config Location	Parameter (Hierarchy)	Description
sping.boot.admin.client.enable:url	Set the admin server url		
sping.boot.application.name	Name of the application		
Solace	application.yml	sping.cloud.stream.binding.input-0.destination	LVQ queue name
sping.cloud.stream.binding.output-0.destination	LVQ topic name		
sping.cloud.stream.binding.output-0.content-type	Message content typ: plain text is application/x-java-serialized-object		

7.2 File Naming Convention

File Type	Naming Convention	Description	Note
Spring Application	application.yml	A standard spring boot configuration	Follow the sring standard yaml notations
Connector for DB source(Config)	connnector_config.properties	Solace DB Connector Configuration file	Generated
Connector For DB souce(Mapper File)	<DB_TABLE>_trgt_schema_mapper.properties </DB_TABLE>	A mapper file which will have the mapping definition between the parent table Schema to the target event payload schema	DB_TABLE value can be identified from connector_config.properties file. All the field mappings should follow the xpath format
Connector for DB source(Mapper File)	<child table="">_trgt_schema_mapper.properties</child>	A mapper file which will have the mapping definition between the parent table Schema to the target event payload schema	Generated

License

This project is licensed under the Apache License, Version 2.0. - 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.