# pubsubplus-connector-couchbase

## *User Guide*

Solace Corporation

Version 2.0.0

solace.

# Table of Contents

# Preface

Solace Couchbase 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 SolaceCouchbaseConnector 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 SolaceCouchbaseConnector producer destination, `producer-destination`.
- Workflow 1 is consuming messages from the SolaceCouchbaseConnector 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
- SolaceCouchbaseConnector

## 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-couchbase-2.0.0.jar --spring.config.additional-
location=file:samples/config/
```

By default, this command detects any Spring Boot configuration files as per the Spring Boot's default locations.

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

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

For convenience, you can start the connector through the shell script using the following command:

```
chmod 744 ./bin/start.sh
./bin/start.sh [-n NAME] [-l FOLDER] [-p PROFILE] [-c FOLDER] [-ch HOST] [-cp PORT] [
-j FILE] [-cm] [-cmh HOST] [-cmp PORT] [-mh HOST] [-mp PORT] [-o OPTIONS] [-b]
```

The script shows you all errors at the same time:

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

The script shows you all errors at the same time:

```
pubsubplus-connector-couchbase

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 situations where you have don't provide a parameter, the script runs with the predefined values as follows:

| Parameter | Default Value | Description |
|---|---|---|
| `-n, --name` | `application` | The name of the connector instance, that is configured in [spring.application.name]. This name impacts on grouping connectors only. |
| `-l, --libs` | `./libs` | The directory that contains the required and optional dependency JAR files, such as Micrometer metrics export dependencies (if configured). If this option is not specified, it will use the current `./libs/` directory. |
| `-p, --profile` | `empty, no profile is used` | PThe profile to be used with the connector's configuration. The configuration file named 'application-<profile>.yml' is used. If this option is not specified, no profile is used. |

| Parameter | Default Value | Description |
|---|---|---|
| `-c, --config` | `./ or current folder` | The path to the folder containing the configuration files to be applied when the connector starts up the chosen profile. If not specified, the current directory is used. |
| `-H, --host` | `127.0.0.1` | Specifies the host where the connector runs. |
| `-P, --port` | `8090` | Specifies the port where connector runs. |
| `-mp, --mgmt_port` | `9009` | Specifies the management port for back calls of current connector from PubSub+ Connector Manager. This parameter is ignored if the `-cm` parameter is not provided. |
| `-j, --jar` | `pubsubplus-connector-couchbase-2.0.0.jar` | The path to the specified JAR file to start the connector. If the option is not specified, the default JAR file is used from the current directory. |
| `-cm, --manager` | `application` | Specifies PubSub+ Connector Manager to use the configuration storage and allows you to enable the cloud configuration for the connector. When this parameter is enabled, you can specify the `-mp` or `--mgmt_port`, `-H` or `--host`, and `-cmh` with the `-cmp` parameters, unless you want to use default values for those parameters. Be aware, this option disable listed parameters to be read from configuration file. In this case, the operator must explicitly specify the parameters for the script, otherwise defaultdefault values are used. |
| `-cmh, --cm_host` | `127.0.0.1` | Specifies the host where Connector Manager is running. This parameter is ignored if the `-cm` parameter is not provided. |

| Parameter | Default Value | Description |
|---|---|---|
| `-cmp, --cm_port` | `9500` | Specifies the port where Connector Manager is running. This parameter is ignored if `-cm` parameter is not provided. |
| `-o, --options` | `no default values` | Specifies the JVM options used on when the connector starts. For example, `-Xms64M -Xmx1G`. |
| `-tls` | `N/A` | Specifies to use HTTPS instead of HTTP. . When this parameter is used, the configuration file must contain an additional section with the preconfigured paths for the key store and trust store files. |
| `-s, --show` | `N/A` | Performs a dry run (does nothing). The output prints the start CLI command and its raw output and exits. This parameter is useful to check your parameters without running the connector. |
| `-b, --background` | `N/A` | Runs the connector in the background. No logs are shown and the connector continues running in detached mode. |
| `-h, --help` | `N/A` | Prints the help information and exits. |

Script also provides that help information from command line using parameter `-h`.

More configuration example of starting Connector together with Connector Manager are provided by the Connector Manager samples.

# Quick Start: Running the connector as a Container

The following steps show how to use the sample docker compose file that has been included in the package:

1. Change to the `docker` directory:

```
cd samples/docker
```

   This directory contains both the `docker-compose.yml` file as well as an `.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:

- Exposes the connector's `8090` web port to `8090` on the host.
- Connects a PubSub+ event broker and SolaceCouchbaseConnector exposed on the host using default ports.
- Mounts the `samples/config` directory.
- Mounts the previously defined `libs` directory.
- Creates a `healthcheck` user with read-only permissions.
  - The default username and password for this user can be found within the `.env` file.
  - This user overrides any users you have defined in your `application.yml`. See here for more information.
- Uses the connector's management health endpoint as the container's health check.

For more information 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-couchbase) # Input system
        output-<workflow-id>:
          destination: <output-destination> # Topic name
          binder: (solace|solace-couchbase) # Output system

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

The connector only supports workflows in the directions of:

- `solace` → `SolaceCouchbaseConnector`
- `SolaceCouchbaseConnector` → `solace`

For more information about Spring Cloud Stream and the Solace PubSub+ binder, see:

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

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

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

For more information 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. For example, 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 example:

```
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-couchbase binder
        solace-couchbase1:
          type: solace-couchbase
          # 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-couchbase1
        output-0:
          destination: <output-destination>
          binder: solace1 # Reference 1st solace binder
        input-1:
          destination: <input-destination>
          binder: solace-couchbase1
        output-1:
          destination: <output-destination>
          binder: solace2 # Reference 2nd solace binder
```

The configuration above defines two binders of type `solace` and one binder of type `solace-couchbase`, which are then referenced within bindings.

Each binder above 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. For example, under the `spring.cloud.stream.binders.<binder-name>.environment`.
- Do not use single-binder configuration (for example, `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 can 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 information about Spring Expression Language (SpEL) expressions, see 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 the future.

## 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 the available registered functions:

| Registered Function Signature | Description |
|---|---|
| `boolean isPayloadBytes(Object obj)` | Returns whether the object `obj` is an instance of `byte[]` or not.<br><br>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 information about Spring Expression Language (SpEL) expressions, see Spring Expression Language (SpEL).

# Message Headers

Solace and solace-couchbase 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 in 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 future releases of this 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.

An Actuator shares information through the endpoints reachable over HTTP/HTTPS. The endpoints that are available are configured in the connector configuration file.

What endpoints are available is configured in the connector configuration file:

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

The above sample configuration enables metrics collection through the configuration parameter of `management.simple.metrics.export.enabled` set to `true` and then shares them through the HTTP/HTTPS endpoint together with other sections configured for the current connector.

### Exposed HTTP/HTTPS Endpoints

The set of endpoints exposed through the HTTP/HTTPS endpoint.

- Exposed endpoints are available if you query the endpoints using the web interface (for example `https://localhost:8090/actuator/<some_endpoint>`) and also available in PubSub+ Connector Manager.
- The operator may choose to not expose all or some of these endpoints. If so, the Actuator endpoints that are not exposed are not visible if you query the endpoints (for example, `https://localhost:8090/actuator/<some_endpoint>`) nor in 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 (for example, Datadog, Prometheus, etc.) to collect metrics.

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

```
GET: /actuator/
```

The following shows an 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 using the Spring Boot Actuator health endpoint.

To configure the information returned by the `health` endpoint, configure the following properties:

- `management.endpoint.health.show-details`
- `management.endpoint.health.show-components`

For more information, about health endpoints, see Spring Boot documentation.

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

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

This configuration would always show the full details 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` | A status that indicates the workflow is functioning as expected. |
| `DOWN` | A status that indicates the workflow is unhealthy. Operator intervention may be required. |

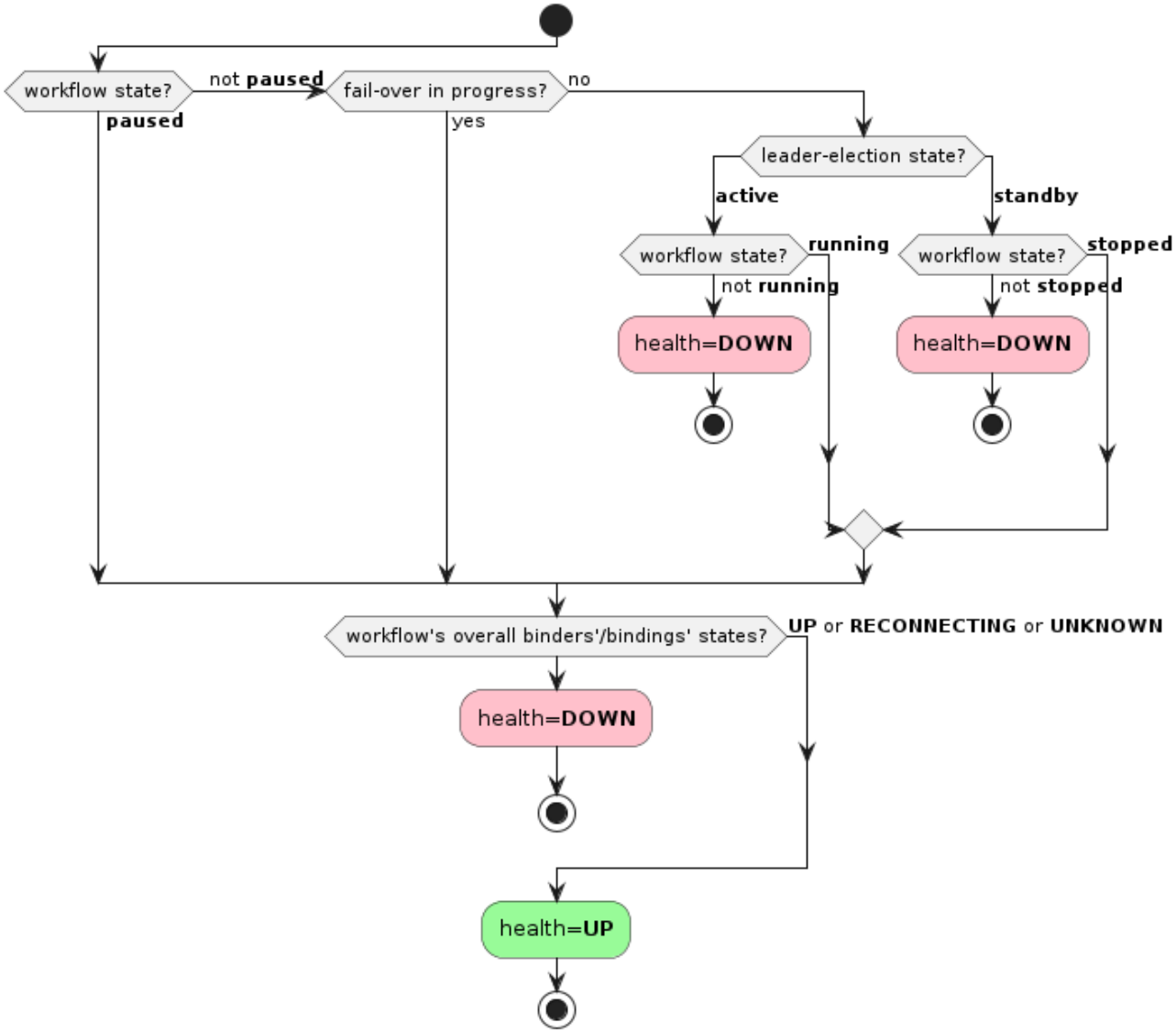



*Figure 1. Workflow Health Resolution Diagram*

This health indicator is enabled default. To disable it, set the property as follows:

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

# Solace Binder Health

For details, see the Solace binder documentation.

# Leader Election

The connector has three leader election modes for redundancy:

| 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 the following configuration:

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

## Leader Election Modes: Standalone / Active-Active

When the connector starts, all enabled workflows start at the same time. The connector itself is considered as always active.

## Leader Election Mode: Active-Standby

If the connector is in active-standby mode, 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 creates 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, failover only happens when there are infrastructure failures (for example, the JVM goes down or networking failures to the management queue).

IIf a workflow fails to start or stop during failover, it will retry up to some maximum defined by the configuration option, `solace.connector.management.leader-election.fail-over.max-attempts`.

During failover, the connector attempts to start or stop all enabled workflows. After an attempt has been made to start or stop each workflow, the connector transitions to the active/standby mode regardless of the status of the workflows.

# Leader Election Management Endpoint

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

Operators can navigate to the connector's `leaderelection` management endpoint to view its leader election status.

| Endpoint | Operation | Payloads |
|---|---|---|
| `/leaderelection` | Read (HTTP `GET`) | **Request:** None.<br><br>**Response:**<br><br>`{ "mode": { "type": "(standalone \| active_active \| active_standby)", ①  "state": "(active \| standby)", ②  "source": { ③  "queue": "<management-queue-name>", "host": "<management-host>", "msgVpn": "<management-vpn>" } } }`<br><br>① Mandatory parameter in output<br><br>② Mandatory parameter in output<br><br>③ Optional section. Appears only when `type` is set to `active_standby`. |

# Workflow Management

## Workflow Management Endpoint

A custom `workflows` management endpoint using Spring Actuator is 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.<br><br>**Response:**<br>Same payload as the `/workflows/{workflowId}` read operation, but as a list of all workflows. |
| `/workflows/{workflowId}` | Read (HTTP `GET`) | **Request:** None.<br><br>**Response:**<br><br>`{ "id": "<workflowId>", "enabled": (true\|false), "state": "(running\|stopped\|paused\|unknown)", "inputBindings": [ "<input-binding>" ], "outputBindings": [ "<output-binding>" ] }` |
| `/workflows/{workflowId}` | Write (HTTP `POST`) | **Request:**<br><br>`{ "state": "STARTED\|STOPPED\|PAUSED\|RESUMED" }`<br><br>**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 continue to persist when write operations are used to change the 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 failover 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. |

When the producer or consumer binding is not implementing Spring's Lifecycle interface, Spring always reports the bindings as `state=N/A`. The `state=N/A` is ignored when deciding the overall state of the workflow. For example, if the consumer's binding is `state=running` and producer's binding `state=N/A` (or vise-versa), the workflow state would be `running`.

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

# Metrics

This connector uses Spring Boot Metrics that 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`<br><br>`name: <bindingName>`<br><br>`result: (success\|failure)`<br><br>`exception: (none\|exception simple class name)` | The 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`<br><br>`name: <bindingNames>`<br><br>`result: (success\|failure)`<br><br>`exception: (none\|exception simple class name)` | The 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`).<br><br>Meters might be merged under the same `name` tag (delimited by `\|`) if multiple bindings have the same error channel name (for example, bindings can have a matching `destination`, `group`, or both). **NOTE: Setting a binding's `group` is not supported.** |
| `solace.connector.message.size.payload` | `DistributionSummary`<br><br>Base Units: `bytes` | `name: <bindingName>` | The message payload size. | |

| Name | Type | Tags | Description | Notes |
|---|---|---|---|---|
| solace.connector.message.size.total | DistributionSummary<br><br>Base Units: bytes | name: <bindingName> | The total message size. | |
| solace.connector.publish.ack | Counter<br><br>Base Units: acknowledgments | name: <bindingName><br><br>result: (success\|failure)<br><br>exception: (none\|exception simple class name) | The 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, we recommend that you 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 includes the following monitoring systems:

- Datadog
- Dynatrace
- Influx
- JMX
- OpenTelemetry (OTLP)
- StatsD

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

# Security

## Securing Endpoints

### Exposed Management Web Endpoints

There are many endpoints that are automatically enabled for this connector. For a comprehensive list, see Management and Monitoring Connector.

The `health` endpoint only returns the root status by default (i.e. no health details).

To enable other management endpoints, see Spring Actuator Endpoints.

### Authentication & Authorization

This release of the connector only supports basic HTTP authentication.

By default, no users are created unless the operator configures them in their configuration file. The configuration parameters responsible for security are as follows:

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

In the above example, we have created two 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 `solace.connector.security.enabled` parameter `false`.

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

The following example shows you 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 two 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, and so on), overriding works by replacing the entire list. In other words, you must pick one place to define all your users, whehter in a **single** application properties file or as environment variables.

For more information, see Spring Boot - Merging Complex Types.

## 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

For information about about how the connector detects configuration properties, see Spring Boot: Externalized Configuration.

### Converting Canonical Spring Property Names to Environment Variables

For information about converting the Spring property names to environment variables, see the Spring documentation.

### Spring Profiles

If multiple configuration files exist within the same configuration directory for use in different environments (development, production, etc.), use Spring profiles.

Using Spring profiles allow you to define different application property files under the same directory using the filename format, `application-{profile}.yml`.

For example:

- `application.yml`: The properties in non-specific files that always apply. Its properties are overridden by the properties defined in profile-specific files.
- `application-dev.yml`: Defines properties specific to the development environment.
- `application-prod.yml`: Defines properties specific to the production environment.

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

See Spring Boot: Profile-Specific Files for more information and an example.

### Configure Locations to Find Spring Property Files

By default, the connector detects any Spring property files as described in the Spring Boot's default locations.

- If you want to add additional locations, add `--spring.config.additional-location=file:<custom-config-dir>` (This parameter is 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>`.

For more information about configuring locations to find Sprint property files, see Spring Boot documentation.

If you want configuration files for multiple, different connectors within the same `config` directory for use in different environments (such as development, production, etc.), we recommend that you use Spring Boot Profiles instead of child directories. For example:

- Set up your configuration like this:
  - `config/application-prod.yml`
  - `config/application-dev.yml`
- Do not do this:
  - `config/prod/application.yml`
  - `config/dev/application.yml`

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

### Obtaining Build Information

Build information, including version, build date, time and description is enabled by default via Spring Boot Actuator Info Endpoint. By default, every connector shares all information related to its build only.

Below is the structure of the output data:

```
{
  "build": {
    "version": "<connector version>",
    "artifact": "<connector artifact>",
    "name": "<connector name>",
    "time": "<connector build time>",
    "group": "<connector group>",
    "description": "<connector description>",
    "support": "<support information>"
  }
}
```

If you want to exclude build data from the output of the `info` endpoint, set `management.info.build.enabled` to `false`.

Alternatively, if you want to disable the info endpoint entirely, you can remove 'info' from the list of endpoints specified in `management.endpoints.web.exposure.include`.

## Spring Configuration Options

This connector packages many libraries for you 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

he following table lists the configuration options. The following options in **Config Option** are prefixed with `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. |
| `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. |

| Config Option | Type | Valid Values | Default Value | Description |
|---|---|---|---|---|
| management.leader-election.mode | enum | (standalone\|active_active\|active_standby) | standalone | The connector's leader election mode.<br><br>**standalone:**<br>A single instance of a connector without any leader election capabilities.<br><br>**active_active:**<br>A participant in a cluster of connector instances where all instances are active.<br><br>**active_standby:**<br>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.*.<br><br>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. |
| security.users[<index>].name | string | any | null | The name of the user. |
| security.users[<index>].password | string | any | null | The password for the user. |
| security.users[<index>].roles | list<string> | admin | empty list (i.e. read-only) | The list of roles that the specified user has. It has read-only access if no roles are returned. |

# Workflow Configuration Options

These configuration options are defined under the following prefixes:

- `solace.connector.workflows.<workflow-id>.`: If the options support per-workflow configuration and the default prefixes.
- `solace.connector.default.workflow.`: If the options support default workflow configuration.

| 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.<br><br>**Value:** A SpEL string that accepts `headers` as parameters. | `empty map` | A mapping of header names to header value SpEL expressions.<br><br>The SpEL context contains the `headers` parameter that can be used to read the input message's headers. |
| `acknowledgment.publish-async` | Per-Workflow Default | `boolean` | `(true\|false)` | `false` | If `true`, publisher acknowledgment processing is done asynchronously.<br><br>The workflow's consumer and producer bindings must support this mode, otherwise the publisher acknowledgments are processed synchronously regardless of this setting. |

| Config Option | Applicable Scopes | Type | Valid Values | Default Value | Description |
|---|---|---|---|---|---|
| acknowledgment.back-pressure-threshold | Per-Workflow Default | int | >= 1 | 255 | The maximum number of outstanding messages with unresolved acknowledgments. Message consumption is paused when the threshold is reached to allow for producer acknowledgments to catch up. |
| acknowledgment.publish-timeout | Per-Workflow Default | int | >= -1 | 600000 | The maximum amount of time (in millisecond) to wait for asynchronous publisher acknowledgments before considering a message as failed. A value of -1 means to wait indefinitely for publisher acknowledgments. |

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

# Couchbase Source Configuration Options

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

| Config Option | Type | Valid Values | Default Value | Description |
|---|---|---|---|---|
| datasource.couchbase.dcpSeedNodes | map | <IP>:<PORT> or <DOMAIN>:<PORT> | | couchbase connection seeds |
| datasource.couchbase.username | string | | | couchbase connection username |
| datasource.couchbase.password | string | | | couchbase connection password |
| datasource.couchbase.bucket | string | | | couchbase connection bucket |

| Config Option | Type | Valid Values | Default Value | Description |
|---|---|---|---|---|
| datasource.couchbase.bootstrapTimeout | int | | 30 | couchbase connectTimeout, in seconds |
| datasource.couchbase.persistencePollingInterval | int | | 2 | couchbase rollback mitigation interval, in seconds |
| datasource.couchbase.flowControlBufferInBytes | long | | 2 | bytes of couchbase flow control buffer size |
| datasource.couchbase.enable-tls | boolean | true or false | false | boolean value of enable-tls |
| datasource.couchbase.trustCertificatePath | string | | | the path of couchbase TrustStore file, must be provided when enable-tls as true |
| source.batchBufferSize | int | | 100 | the batch buffer size of events send to solace |
| source.pollingInterval | int | | 2000 | the interval milliseconds of polling from couchbase |
| source.bindings | map | | | map of embedding configurations of bindings |
| source.bindings.<input-x>.scope | string | | | the scope polling from couchbase |
| source.bindings.<input-x>.collections | string | | | the collections polling from couchbase |
| source.bindings.<input-x>.topicPattern | string | | | customize topic by pattern, available pattern keys: BUCKET,DOCUMENT_ID,PARTITION,OFFSET_UUID,SEQ_NO,CAS,SCOPE,SCOPE_ID,COLLECTION,COLLECTION_ID |
| source.bindings.<input-x>.lvq | string | | | the lvq to hold offset status of polling |
| source.bindings.<input-x>.lvqTopic | string | | | the topic of lvq |
| source.bindings.<input-x>.couchbaseMetaData | embedding | | | embedding configuration of couchbase metadata usage |

These configuration options are all prefixed by `solace-persistence.source.bindings.<input-x>.couchbaseMetaData`:

| Config Option | Type | Valid Values | Default Value | Description |
|---|---|---|---|---|
| enabled | boolean | true or false | false | if true, couchbase metadata would be extracted to solace message |
| documentId | string | | | document id of couchbase document will be presented to specified json node of payload |
| metaIncorporateIn | string | header or payload | header | where to place the extracted metadata |
| customMeta | list | | | list of KV pairs of extracted metadata |
| customMeta.headerKeyName | string | | | header name |
| customMeta.headerKeyValue | string | | | metadata key of couchbase document, available meta key: qualifiedKey, key, cas, revision, tracingToken, timestamp, vbucket, isMutation,offset.seqno, offset.vbuuid, collection.id, collection.name, scope.id, scope.name |

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

# Couchbase Sink Configuration Options

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

| Config Option | Type | Valid Values | Default Value | Description |
|---|---|---|---|---|
| datasource.couchbase.dcpSeedNodes | map | <IP>:<PORT> or <DOMAIN>:<PORT> | | couchbase connection seeds |
| datasource.couchbase.username | string | | | couchbase connection username |
| datasource.couchbase.password | string | | | couchbase connection password |
| datasource.couchbase.bucket | string | | | couchbase connection bucket |

| Config Option | Type | Valid Values | Default Value | Description |
|---|---|---|---|---|
| datasource.couchbase.bootstrapTimeout | int | | 30 | couchbase connectTimeout, in seconds |
| datasource.couchbase.persistencePollingInterval | int | | 2 | couchbase rollback mitigation interval, in seconds |
| datasource.couchbase.flowControlBufferInBytes | long | | 2 | bytes of couchbase flow control buffer size |
| datasource.couchbase.enable-tls | boolean | true or false | false | boolean value of enable-tls |
| datasource.couchbase.trustCertificatePath | string | | | the path of couchbase TrustStore file, must be provided when enable-tls as true |
| sink.topic2Collection | string | <topic>:<scope>.<collection>,… | | mappings of solace topic and couchbase scope & collection relationship |
| sink.bindings | map | | | map of embedding configurations of bindings |
| sink.bindings.<output-x>.payloadNode | string | | | the json node prefix of payload |
| sink.bindings.<output-x>.documentId | string | | | node of documentId in payload, starts with "/" |
| sink.bindings.<output-x>.removeDocumentId | boolean | true or false | true | when true, the node of documentId will not be presented in couchbase content |
| sink.bindings.<output-x>.expireSeconds | int | | | couchbase content expire time, in seconds |

# More information

## Solace Pubsub Connector For Database Sink

### 1 Introduction

The Solace Pubsub+ Connector For Couchbase Source provides integration between PubSub+ event brokers/mesh and Couchbase, it helps to read the Couchbase Scope/collection data and make it available as an event in the Solace Event Mesh for enterprise applications to consume.

The Solace Pubsub+ Connector For Couchbase Sink will consume the event from Solace Broker/mesh and insert/update to the Couchbase Scope/collection.

The Solace Pubsub+ Connector For Couchbase is based on the new common architecture of the Solace Framework Connectors

#### 1.1 Pre-requisites

##### 1.1.1 Couchbase

- Couchbase cluster 7.2.3

##### 1.1.2 Java

- Solace Pubsub+ Connector For Couchbase required java runtime.
- Install jdk/jre 17.0.8.

##### 1.1.3 Maven

- Apache Maven 3.9+

### 2 Resource Requirement

Below given is the minimum viable production deployment resource numbers. Solution can be scaled horizontally with clustered mechanism and resources can be allocated accordingly.

- 4 Core CPU 4 GB.
- Solace 10 GB.

### 3 Broker detail

The Solace Broker manager address: `http://SolaceIP:8080/`.

In the application.yml file, it is configured to this Broker.

```
solace:
  java:
    host: tcp://SolaceIP:Port
```

```
    msgVpn: default
    clientUsername: username
    clientPassword: password
    connectRetries: -1
    reconnectRetries: -1
    apiProperties:
      pub_ack_window_size: 50
      pub_ack_time: 200
```

## 4 Couchbase Connection Detail

The Couchbase connection is configured as below in the application.yml

```
solace-persistence:
  datasource:
    couchbase:
      dcpSeedNodes:
        - IP:PORT
      username: username
      password: password
      bucket: bucket-name
      bootstrapTimeout: 30   #  seconds
      persistencePollingInterval: 2  #  seconds
      flowControlBufferInBytes: 10485760 # 10M
      enable-tls: true
      trustCertificatePath: path-to-tuststore-file
```

## 5 Scenarios

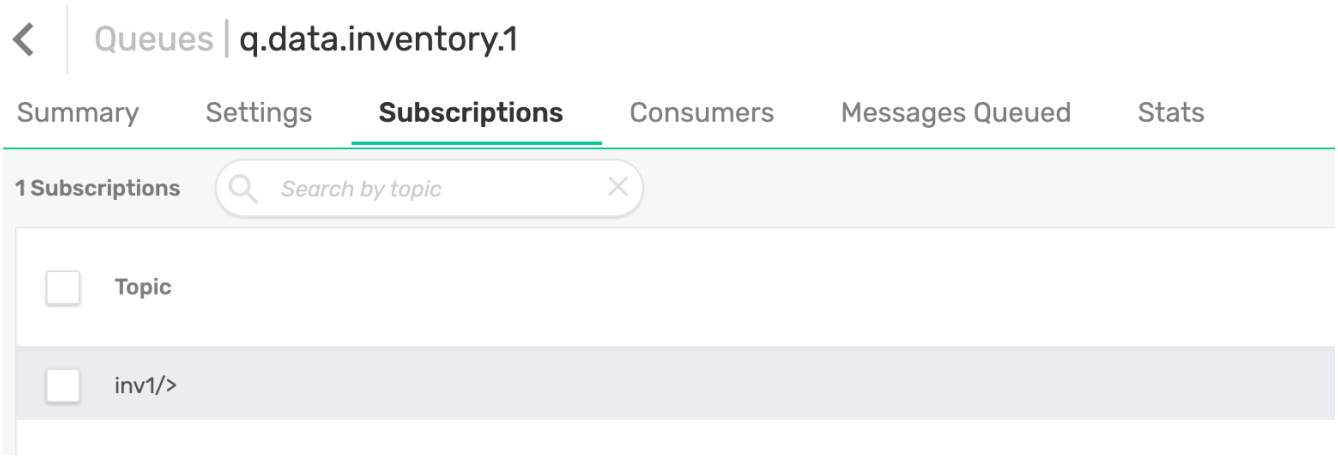
### 5.1 Couchbase to Solace Topic/Queue

- Couchbase connection, bucket, scope and collection.

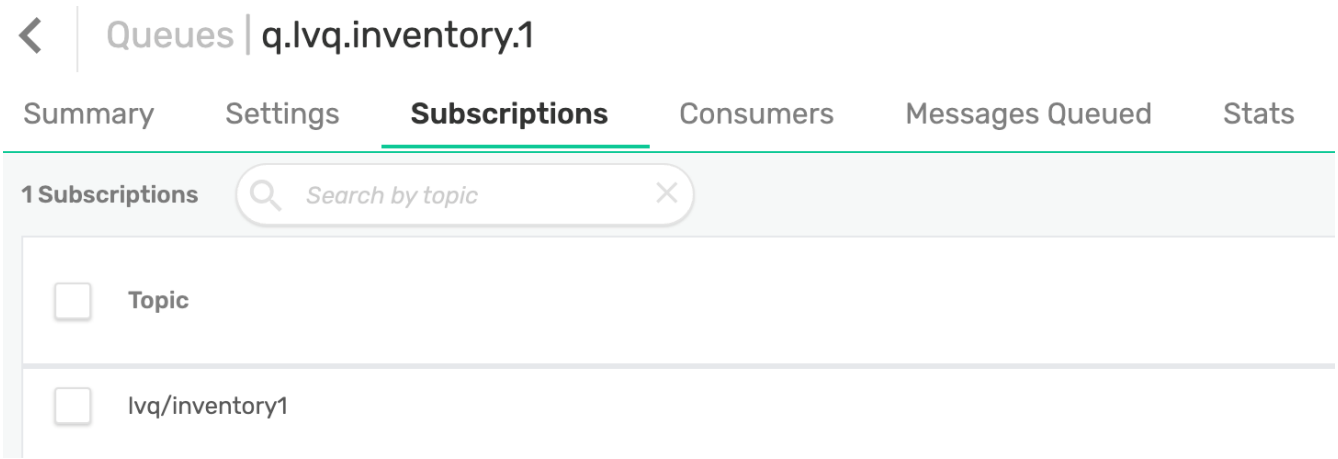
```
solace-persistence:
  datasource:
    couchbase:
      dcpSeedNodes:
        - IP:PORT
      username: username
      password: password
      bucket: bucket-name
      bootstrapTimeout: 30   #  seconds
      persistencePollingInterval: 2  #  seconds
      flowControlBufferInBytes: 10485760 # 10M
      enable-tls: true
      trustCertificatePath: path-to-tuststore-file
  source:
    bindings:
```

```
      input-0:  # workflow 0
        scope: scope
        collections: collections
```

- Queue name: q.data.inventory.1, it has subscription to topic inv1/>, the connector will read the couchbase data as payload and send to this topic/Queue.




- Queue name: q.lvq.inventory.1, it has subscription to topic lvq/inventory, the connector will send the offset data this topic/Queue.




- Please see the configuration of the data topic, LVQ topic and LVQ

```
    input-1:           #workflow 1
      scope: inventory
      collections: hotel
      couchbaseMetaData:
        enabled: true        #if ture, it will add metadata to the header or payload of the Solace message
        documentId: _id_     #add documentId in Payload, in the payloade it use _id_ as the element name
        metaIncorporateIn: payload # header or payload
        customMeta:
          # available meta key :
          # qualifiedKey, key, cas, revision, tracingToken, timestamp, vbucket, isMutation,
          # offset.seqno, offset.vbuuid, collection.id, collection.name,  scope.id, scope.name
          - headerKeyName: MESSAGE_HEADER_KEY
            headerKeyValue: key
          - headerKeyName: MESSAGE_HEADER_COLLECTION_NAME
            headerKeyValue: collection.name
          - headerKeyName: MESSAGE_HEADER_COLLECTION_ID
            headerKeyValue: collection.id
      topicPattern: inv1/{SCOPE}/{COLLECTION}/{BUCKET}/{DOCUMENT_ID}
      lvq: q.lvq.inventory.1
      lvqTopic: lvq/inventory1
```

- Metadata configuration see the details

```
    input-1:           #workflow 1
      scope: inventory
      collections: hotel
      couchbaseMetaData:
        enabled: true        #if ture, it will add metadata to the header or payload of the Solace message
        documentId: _id_     #add documentId in Payload, in the payloade it use _id_ as the element name
        metaIncorporateIn: payload # header or payload
        customMeta:
          # available meta key :
          # qualifiedKey, key, cas, revision, tracingToken, timestamp, vbucket, isMutation,
          # offset.seqno, offset.vbuuid, collection.id, collection.name,  scope.id, scope.name
          - headerKeyName: MESSAGE_HEADER_KEY
            headerKeyValue: key
          - headerKeyName: MESSAGE_HEADER_COLLECTION_NAME
            headerKeyValue: collection.name
          - headerKeyName: MESSAGE_HEADER_COLLECTION_ID
            headerKeyValue: collection.id
      topicPattern: inv1/{SCOPE}/{COLLECTION}/{BUCKET}/{DOCUMENT_ID}
      lvq: q.lvq.inventory.1
      lvqTopic: lvq/inventory1
```

- In this configuration, workflow 0 is disabled and it will not work, workflow 1 is enabled and will read data from Couchbase bucket/inventory/collection

```
solace:
  connector:
    workflows:  # Workflow configuration
      0:
        enabled: false  # If true, the workflow is enabled.
      1:
        enabled: true  # If true, the workflow is enabled.
```

### 5.2 Solace Queue to Couchbase

- The Source Data queue is q.data.inventory.1, see the sample configuration of Solace Queue.

```
    stream:
      bindings:  # Workflow bindings
        input-0:
          destination: q.data.inventory.1   # Queue name
          binder: solace
          consumer:
            concurrency: 1
            batchMode: true
        output-0:
          destination: scope.collection  # SolaceCouchbaseConnector producer destination
          binder: solace-couchbase
```

- The Couchbase configuration, see the sample configuration.

```
solace-persistence:
  datasource:
    couchbase:
      dcpSeedNodes:
        - IP:PORT
      username: username
      password: password
      bucket: bucket-name
      bootstrapTimeout: 30   #  seconds
      persistencePollingInterval: 2  #  seconds
      flowControlBufferInBytes: 10485760 # 10M
      enable-tls: true
      trustCertificatePath: path-to-tuststore-file
  sink:
    topic2Collection: topic1:tenant_agent_00.bookings,
    bindings:
      output-0:
        payloadNode:
        documentId: /_id_
        removeDocumentId: true
        expireSeconds: 30
```

Configure the topic to scope/collection mapping, it read all the data with topic name start with inv1 to the tenant_agent_00.bookings1

`documentId` is set to `/_id_`, the connector will use the element `_id_` of the payload as the documentId in Couchbase collection. `removeDocumentId: ture`, it will remove it from the payload in the collection.

- configure the workflows and enable it

```
46  solace:
47    connector:
48      workflows:  # Workflow configuration
49        0:
50          enabled: false  # If true, the workflow is enabled.
51        1:
52          enabled: true  # If true, the workflow is enabled.
```

# 6 Run Connector

### 6.1Solace Connector for Couchbase Source

```
java -cp ./target/pubsubplus-connector-couchbase-1.0.0-SNAPSHOT.jar
org.springframework.boot.loader.PropertiesLauncher --spring.config.additional-
location=./src/main/resources/couchbase-source/
```

### 6.2 Solace Connector for Couchbase Sink

```
java -cp ./target/pubsubplus-connector-couchbase-1.0.0-SNAPSHOT.jar
org.springframework.boot.loader.PropertiesLauncher --spring.config.additional-
location=./src/main/resources/couchbase-sink/
```

# 7 Tools

### 7.1 JMSToolbox

JMSToolBox is a JMS client based on the Eclipse platform, it is a Free Open Source Software (FOSS).

You can add a session to Solace:

Update Session

Connection | Properties

Definition

Session Name: 54.157.145.242

Queue Manager: Solace

Folder:

Session Type: Production | Clear

Servers

Host / Port: 54.157.145.242 : 55555

Host / Port (2):

Host / Port (3):

Security

User id: cbuser

Password: ••••••

☐ Prompt for userid/password

⚠ If the userid/password are NOT saved in the session, the 'REST' and 'scripts' features of JMSToolBox will not work

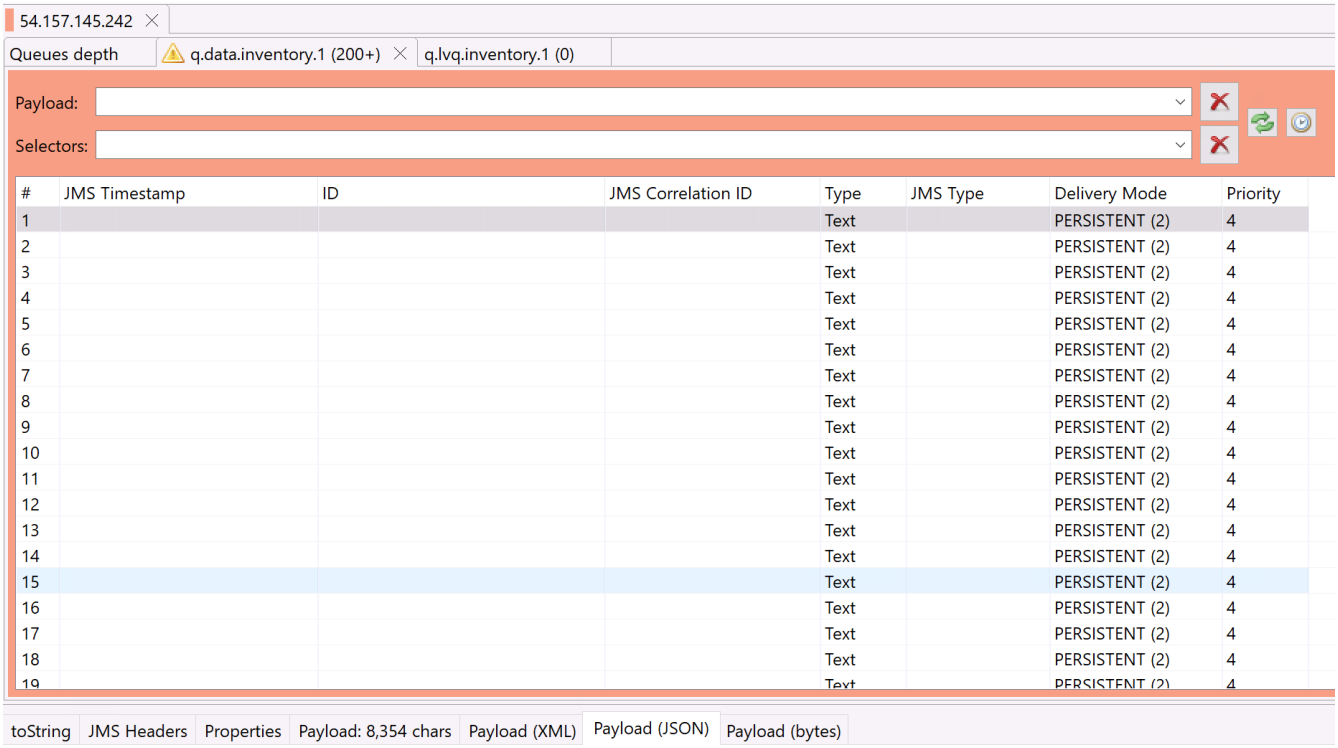
Update Session

Connection | Properties

| | Name | Kind | Value |
|---|---|---|---|
| * | VPN | STRING | couchbaseVPN |
| * | browser_timeout | INT | 250 |
| * | mgmt_password | STRING | cbuser |
| * | mgmt_url | STRING | http://54.157.145.242:8080 |
| * | mgmt_username | STRING | cbuser |
| | ssl_cipher_suite | STRING | |
| | ssl_connection_downgrade_to | STRING | |
| | ssl_excluded_protocols | STRING | |
| | ssl_key_store | STRING | |
| | ssl_key_store_format | STRING | |
| | ssl_key_store_password | STRING | |
| | ssl_private_key_alias | STRING | |
| | ssl_private_key_password | STRING | |
| | ssl_protocol | STRING | |
| | ssl_trust_store | STRING | |
| | ssl_trust_store_format | STRING | |
| | ssl_trust_store_password | STRING | |
| | ssl_trusted_common_name_list | STRING | |
| | ssl_validate_certificate | BOOLEAN | |
| | ssl_validate_certificate_date | BOOLEAN | |

Update | Cancel

Browse the queue message information using jmstoolbox.





```
{
   "metadata":{
      "MESSAGE_HEADER_KEY":"hotel_16447",
      "MESSAGE_HEADER_COLLECTION_NAME":"hotel",
      "MESSAGE_HEADER_COLLECTION_ID":"20"
   },
   "payload":{
      "country":"United Kingdom",
      "free_parking":true,
      "city":"London",
      "description":"4 Star hotel near Kensington High Street with 906 rooms offering the usual Holiday Inn services",
      "_id_":"hotel_16447",
      "title":"London/South Kensington-Chelsea",
      "type":"hotel",
      "geo":{
         "lat":51.4942,
```


Please check the details in https://github.com/jmstoolbox/jmstoolbox

### 7.2 SDKperf

Another option for browsing queues in Solace, if you want something really basic (i.e. command-line text dump) is to use the SdkPerf test tool, which you download in different flavours from our solace.com/downloads page.

Run it like so (C for Linux example given, but Java/JMS/C# similar):

This makes the messages appear every 5 seconds. Just delete this if you want thing to scroll quickly.

```
./sdkperf_c -cip= 54.157.145.242 -cu=cbuser@couchabseVPN -cp=cbuser -sql=
q.data.inventory.1  -qb -md
```

```
-cip = broker hostname/IP address
-cu = client-username @ message-vpn (optional, will be default@default if omitted)
```

```
-cp = password (optional)
-sql = which queue to subscribe/connect to
-qb = enable queue browsing (rather than consuming!)
-md = message dump to console
-sd = subscriber delay, per message (optional).
```

be very careful to ensure you have the -qb option enabled, otherwise your SdkPerf instance will consume the messages, rather than browsing them. Which means they get removed from the queue. As always, the SdkPerf command line options -h (help) and -hm (help more) are very useful.

# 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.