Verification Guide for Confluent Platform Integrations

Overview

Confluent Platform is a streaming platform, built on Apache Kafka®. This document describes the criteria that we use in verifying integrations provided by third-parties for streaming data into Confluent Platform from other technologies. A common example of these integrations is Change-Data-Capture (CDC) from relational databases (RDBMS) such as DB2. Other examples of integrations include streaming from Kafka to a target datastore such as Couchbase, or Elasticsearch.

Integrations are typically built using the Kafka Connect API. This provides the easiest way to fully integrate source and target systems with Apache Kafka. The Kafka Connect API was added into Apache Kafka in release 0.9. Integrations built using Kafka Connect are referred to as "Connectors". Partners may chose to implement integrations using the Kafka Producer API, but this is not recommended as Kafka Connect is designed to specifically handle many of the common integration requirements (offset tracking, data serialisation, schema management, etc).

Verified Integrations must meet a **standard** level of functional compatibility with the the Confluent Platform ecosystem. A higher **gold** level requires provision of all required functionality listed in this document. If an integration does not meet the basic level of functionality then it will not be classed as a verified integration by Confluent.

In addition to meeting the verification criteria, a Verified Integration must also be fully supported by the partner.

This guide defines formal connector verification criteria using the MOSCOW principal. Connector verification is divided into two tiers—standard and gold. Connectors that are certified under the standard tier meet all the "must have" criteria listed in this guide. Connectors that also meet all the "should have" criteria are verified under the gold tier.

Table of Contents

General Objectives

Why Verification?

Verifying connectors is a mutually beneficial relationship between Confluent and its partners, as well as their direct customers. On the customer side, the verification of an integration provides assurances of a level of compatibility and functionality with the Confluent Platform ecosystem. There is a large number of community-supported Connectors. By obtaining a “verified” status for their integration, Partners can signal to prospects a status above others. In addition, the ecosystem for Confluent and its partners gains both in diversity and in depth—Kafka is a genuine central hub for all things, and each additional connector exponentially increases the possible interconnects.

What does it mean to be Verified?

Verification is divided into two levels: gold and standard. **Gold** integrations have a high standard in terms of documentation, test coverage, and functionality of integration with Confluent Platform. **Standard** integrations may omit certain functionality, or proven test coverage.

This document lists all conditions of verification using the MOSCOW method. The MOSCOW methodology is well suited to be unambiguous, agile, and straight forward. It is applied to the verification levels thus:

- An integration meeting all **Must** and **Should** categories will have a **gold level** rating
- An integration meeting all **Must** categories will have a **standard level** rating

With regards to “Could” and “Could Not”, these are often suggestions for partners that will make developing the connector easier.

Verification Criteria

The following is a summary of what we will assess. See later in this document for full details.

	Verified: Gold	**Verified: Standard**
Fully supported by Partner	✓	✓
Functionality		
Produce/Consume JSON	✓	✓
Produce/Consume Avro with Schema Registry	✓	✓
Implement Single Message Transforms	✓	
Pluggable Connect converters	✓	
Confluent Control Center interceptors	✓	
Provides JMX Metrics	✓	
Runs within Connect Worker	✓	
Supports Highly-Available deployment model	✓	
Exactly-Once Delivery Guarantees (sink only, where supported by target technology)		
Internals		
Unit Tests (see doc)	✓	✓
Configuration (see doc)	✓	✓
Schema Compatibility - document supported types	✓	✓
Offset Management	✓	✓
Implement graceful back-off operations that require retries.	✓	✓
System Tests	✓	
Schema Migration (Sink only)	✓	
Internals (Kafka Connect only)		
Packaging : Classes/classpath (see doc)	✓	✓
Parallelism	✓	✓
Error Handling	✓	
Packaging : JAR structure (see doc)	✓	

Versioning

Verifications are version sensitive. Submitters MUST clearly define three versions:

1. Connector Version.
2. Compatible Source technology/Sink technology Versions.
3. Compatible Confluent Platform Versions.

The verification for the connectors applies only to the three constraints listed above. Changes to any of them would require further Verification.

The following table is an example of how Confluent will publish verified status:

Connector Name	**Connector Version**	**Source Versions**	**Sink Versions**	**Verification Status**
Confluent JDBC Connector	1.1	PostgreSQL 8.X MySQL 5.X MariaDB 10.X	PostgreSQL 8.X MySQL 5.X MariaDB 10.X	Verified Gold

Connector Name	**Avro with Schema**	**JSON with Schema**	**SMT**	**Pluggable Converters**	**C3 Interceptors**
Confluent JDBC Connector	✓	✓	✓	✓	✓

A Note About Open Source Vs Closed Sourced Connectors

Even though the majority of Connectors in the ecosystem are open source, certain partners may wish to retain the rights to not publish their source code. This is fully acceptable. Careful consideration has been given in this document to allow all "MUST" criteria to be independently verified without looking at source.

Kafka Connect vs Producer APIs

Partners may chose to implement integrations using the Kafka Producer API in lieu of the Connect API, but this is not recommended as Kafka Connect is designed to specifically handle many of the common integration requirements (offset tracking, data serialisation, schema management, etc). In addition, any functionality implemented outside of Kafka Connect must present the same interface to the user. For example:

- Standard configuration options must match those provided by Kafka Connect, e.g. schema.registry.url.

In summary, a developer using the integration must have the same experience using it regardless of whether it was implemented using Kafka Connect or Producer API.

Development Resources

Partners have access to the following resources:

1. Kafka Connect overview, concepts, and architecture
2. Kafka Connect Developer Guide
3. Partner Development Guide for writing Connectors
4. Blog post describing Kafka Connect development
5. Confluent Partner Technology Briefing: Developing Connectors in the Kafka Connect Framework (password available on request for Partner program members)

Verification Submission

Once an integration is ready for verification by Confluent, the process is as follows:

1. Partner must already be a Registered member of the Confluent Partner Program (apply here)
    a. Once an integration has been verified and a status (Standard or Gold) issued the Partner will move to Preferred status in the program.
2. Email partners@confluent.io with the request for verification and include all supporting artifacts.
3. Confluent will verify the integration and provide feedback to the Partner on any issues
4. Prior to issuing a final verification, Confluent will discuss with the Partner the proposed verification status, and give the Partner the opportunity to revise the integration to address any issues raised.

Verification Checklist

Documentation

1. The integration MUST include a quick-start tutorial showing step-by-step how to use the integration. As well as benefiting the end-user, this documentation will be used by Confluent when performing the verification process.
2. The integration COULD include Docker images of the necessary source/target technology to enable easier test environment setup for Confluent when verifying the integration.

Unit Tests

Partner connectors:

1. MUST provide single line CLI command to execute all unit tests.
2. Unit tests MUST print out human readable report on test name and their pass/fail status.
3. Unit tests MUST cover
   - All internal APIs
   - Configuration Validation
   - Data Conversion from Connect to Data Systems
   - Framework Integration

System Tests

1. MUST produce document detailing the test environment, to include details including configuration on source/sink cluster setups, kafka setup.
2. MUST produce report showing proper instantiation of the Connector within Kafka Connect Workers.
3. MUST support schema driven data conversion with both Avro and JSON serialization classes.
4. MUST produce report demonstrating task restart and rebalance in the event of worker node failure.
5. (1) – (3) MUST be demonstrated in distributed cluster mode.
6. (1) – (3) SHOULD be demonstrated in stand-alone mode as well.
7. The above tests SHOULD be delivered to confluent with an automated script that verifiers can run to reproduce all scenarios easily.
8. The above tests COULD be done with the Confluent System Test Framework ("Ducktape").

Connector Configuration

1. MUST include customer facing documentation listing all configurable options.
2. MUST include customer facing documentation on configuration details of each option.
3. MUST be able to do basic syntax check on valid configurations, output properly formatted error on invalid configuration syntax.
4. MUST be able to configure authentication to target source and sinks.
5. SHOULD include configuration validator that can check configuration sanity on inter-dependent options.

Schema Compatibility

1. MUST include customer facing documentation on supported data types and the expected message syntax.
2. SHOULD NOT simply cast fields from incoming messages to the expected data types.
3. Source Connectors SHOULD associate explicit data schemas with their messages.
4. MUST throw appropriate exceptions if the data type is not supported.
5. In (4)'s case, SHOULD use ConnectException instead of ClassCastException.
6. SHOULD support schemaless data.

Schema Migration

1. Sink Connectors MUST keep track of latest schema version of incoming data.
2. Sink Connectors MUST operate with a mix of schema versions.
3. Sink Connectors SHOULD use Schema Projector to project scheme between compatible versions.

Offset Management

Source Connectors

1. Upon executing start(), connectors MUST retrieve the last committed offsets for the configured Kafka topics.
2. Connectors MUST map the committed offsets with their correct counterparts in the source system and include the offset with each record. Upon start(), the connector SHOULD use those committed offset to identify where in the source system to begin reading.

Sink Connectors

1. Developers MUST provide documentation on how partitioned topics are handled.
2. The connector MUST respect correct ordering of data to the target.
3. If exactly once semantics is supported, connectors MUST return the offsets of the messages that have been written to the target and MUST deliver messages exactly one time.

Converters and Serialization

1. Serialization logic MUST be handled separately from the connector.
2. Connectors MUST be compatible with delegating the converting and serializing data to the CONNECT framework.

Parallelism

1. Connector.taskConfigs() MUST be able to divide the work into the requested maximum number of tasks. Connectors MAY choose to use fewer tasks than allowed.

Error Handling

1. Connectors MUST NOT throw RuntimeException unless it encounters a truly fatal error.
2. Connectors SHOULD catch exceptions and either handle them gracefully to continue operation or rethrow them as ConnectException or one of its subtypes.
3. Connectors MUST implement graceful back-off operations that require retires.

Packaging

Partner Connectors:

1. MUST NOT directly include any of the following classes in the connector jar
    - io.confluent.*
    - org.apache,kafka.connect.*
2. MUST NOT conflict with packaged connectors without CLASSPATH isolation.
3. Archive Structure MUST be laid out in logical manner.

4. Archive Structure SHOULD be laid out as such:

```
kafka-connect-<connector>/
├── LICENSE
├── NOTICE
├── README.md
├── config
│   ├── [sample.properties]
├── docs
│   ├── requirements.txt
│   ├── sink_config_options.rst
│   ├── sink_connector.rst
│   ├── source_config_options.rst
│   └── source_connector.rst
├── licenses
│   ├── [bundled_license.txt]
├── pom.xml
├── src
│   ├── assembly
│   │   ├── development.xml
│   │   ├── package.xml
│   │   └── standalone.xml
│   ├── main
│   │   ├── java
│   │   │   └── [com/io/org …]
│   │   │       └── [company_name]
│   │   │           └── connect
│   │   │               └── [Connector Name]
│   │   │                   ├── sink
│   │   │                   │   ├── dialect
│   │   │                   │   └── metadata
│   │   │                   ├── source
│   │   │                   └── util
│   │   │                       └── Version.java
│   │   └── resources
│   │       └── [sample.properties]
│   └── test
│       ├── java
│       │   └── [com/io/org …]
│       │       └── [company_name]
│       │           └── connect
│       │               └── [Connector Name]
│       │                   ├── sink
│       │                   │   ├── dialect
│       │                   │   └── metadata
│       │                   ├── source
│       │                   └── util
│       └── resources
│           └── [sample.properties]
└── version.txt
```

Revision History

Version	When	Who	What
1.50	February 2018	Chong Yan, Robin Moffatt	Revised edition
1.51	15 Mar 2018	Robin Moffatt	Remove requirement for embedded-schema JSON