Red Hat build of Debezium 2.3.4

Release Notes for Red Hat build of Debezium 2.3.4

What’s new in Red Hat build of Debezium
What's new in Red Hat build of Debezium
Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at http://creativecommons.org/licenses/by-sa/3.0/. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation’s permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Describes the Red Hat build of Debezium product and provides the latest details on what’s new in this release.
Table of Contents

CHAPTER 1. RED HAT BUILD OF DEBEZIUM ....................................................... 3

CHAPTER 2. DEBEZIUM 2.3.4 RELEASE NOTES .................................................. 4
  2.1. DEBEZIUM DATABASE CONNECTORS ..................................................... 4
  2.1.1. Connector usage notes ................................................................. 4
  2.2. DEBEZIUM SUPPORTED CONFIGURATIONS .......................................... 5
  2.2.1. AMQ Streams API version ............................................................. 5
  2.3. DEBEZIUM INSTALLATION OPTIONS .................................................. 5
  2.4. UPGRADING DEBEZIUM FROM VERSION 1.X TO 2.3.4 ............................. 5
    2.4.1. Upgrading connectors to Debezium 2.3.4 ...................................... 5
  2.5. NEW FEATURES AND IMPROVEMENTS ............................................... 6
    2.5.1. Breaking changes ................................................................. 7
      2.5.1.1. New configuration defaults for MySQL and PostgreSQL secure connections 7
      2.5.1.2. Topic and schema naming changes ....................................... 7
      2.5.1.3. Changes to the Oracle connector source information block .......... 8
    2.5.2. Features promoted to General Availability ................................... 8
    2.5.3. General availability features .................................................... 8
      2.5.3.1. Automated replica identity configuration for PostgreSQL ........... 9
      2.5.3.2. New notification subsystem ................................................. 9
      2.5.3.3. Correlate incremental snapshot notification IDs ..................... 10
      2.5.3.4. Support for new signaling channels ................................... 10
      2.5.3.5. Oracle RAC improvements .................................................. 11
      2.5.3.6. Oracle connector SCN-based metrics .................................. 11
      2.5.3.7. Server side filtering for the MongoDB and Oracle connectors .... 11
      2.5.3.8. Retry database connections during connector startup .............. 12
      2.5.3.9. Use of surrogate keys in incremental snapshots ..................... 12
      2.5.3.10. ExtractChangedRecordState SMT ....................................... 13
      2.5.3.11. Drop event fields with new configuration options for the ExtractNewRecordState SMT .................................................. 14
    2.5.4. Technology Preview features ..................................................... 15
      2.5.4.1. MongoDB sharded cluster improvements (Technology Preview) ... 15
      2.5.4.2. MongoDB incremental snapshots for multi-replica and sharded clusters (Technology Preview) .................................................. 16
    2.5.5. Developer Preview features ....................................................... 16
      2.5.5.1. JDBC Sink Connector (Developer Preview) ............................. 17
      2.5.5.2. MySQL connector: parallel snapshots (Developer Preview) ....... 18
      2.5.5.3. Oracle connector: Ingesting changes from an Oracle logical standby (Developer Preview) .................................................. 18
      2.5.5.4. Exactly-once delivery for PostgreSQL connector (Developer Preview) .................................................. 18
    2.5.6. Other updates in this release ..................................................... 19
  2.6. DEPRECATED FEATURES ...................................................................... 23
  2.7. KNOWN ISSUES .................................................................................. 23
CHAPTER 1. RED HAT BUILD OF DEBEZIUM

Red Hat build of Debezium is a comprehensive set of integration and event processing technologies for creating, extending, and deploying container-based integration services across hybrid and multicloud environments. Red Hat build of Debezium provides an agile, distributed, and API-centric solution that organizations can use to connect and share data between applications and systems required in a digital world.

Red Hat build of Debezium includes the following capabilities:

- Real-time messaging
- Cross-datacenter message streaming
- API connectivity
- Application connectors
- Enterprise integration patterns
- API management
- Data transformation
- Service composition and orchestration

Additional resources

- Understanding enterprise integration
CHAPTER 2. DEBEZIUM 2.3.4 RELEASE NOTES

Debezium is a distributed change data capture platform that captures row-level changes that occur in database tables and then passes corresponding change event records to Apache Kafka topics. Applications can read these change event streams and access the change events in the order in which they occurred. Debezium is built on Apache Kafka and is deployed and integrated with AMQ Streams on OpenShift Container Platform or on Red Hat Enterprise Linux.

The following topics provide release details:

- Section 2.1, “Debezium database connectors”
- Section 2.2, “Debezium supported configurations”
- Section 2.3, “Debezium installation options”
- Section 2.4, “Upgrading Debezium from version 1.x to 2.3.4”
- Section 2.5, “New features and improvements”
- Section 2.6, “Deprecated features”
- Section 2.7, “Known issues”

2.1. DEBEZIUM DATABASE CONNECTORS

Debezium provides connectors based on Kafka Connect for the following common databases:

- Db2
- JDBC Sink connector (Developer preview)
- MongoDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server

2.1.1. Connector usage notes

- Db2
  - The Debezium Db2 connector does not include the Db2 JDBC driver (jcc-11.5.0.0.jar). See the Db2 connector deployment instructions for information about how to deploy the necessary JDBC driver.
  - The Db2 connector requires the use of the abstract syntax notation (ASN) libraries, which are available as a standard part of Db2 for Linux.
  - To use the ASN libraries, you must have a license for IBM InfoSphere Data Replication (IIDR). You do not have to install IIDR to use the libraries.
Oracle

- The Debezium Oracle connector does not include the Oracle JDBC driver (ojdbc8.jar). See the Oracle connector deployment instructions for information about how to deploy the necessary JDBC driver.

PostgreSQL

- To use the Debezium PostgreSQL connector you must use the pgoutput logical decoding output plug-in, which is the default for PostgreSQL versions 10 and later.

Additional resources

- Getting Started with Debezium
- Debezium User Guide

2.2. DEBEZIUM SUPPORTED CONFIGURATIONS

For information about Debezium supported configurations, including information about supported database versions, see the Debezium 2.3.4 Supported configurations page.

2.2.1. AMQ Streams API version

Debezium runs on AMQ Streams 2.5.

AMQ Streams supports the v1beta2 API version, which updates the schemas of the AMQ Streams custom resources. Older API versions are deprecated. After you upgrade to AMQ Streams 1.7, but before you upgrade to AMQ Streams 1.8 or later, you must upgrade your custom resources to use API version v1beta2.

For more information, see the Debezium User Guide.

2.3. DEBEZIUM INSTALLATION OPTIONS

You can install Debezium with AMQ Streams on OpenShift or on Red Hat Enterprise Linux:

- Installing Debezium on OpenShift
- Installing Debezium on RHEL

2.4. UPGRADING DEBEZIUM FROM VERSION 1.X TO 2.3.4

The current version of Debezium includes changes that require you to follow specific steps when you upgrade from versions earlier than 2.1.4.

2.4.1. Upgrading connectors to Debezium 2.3.4

The Debezium 2.3.4 release includes some changes that are not backward-compatible with versions of Debezium earlier than 2.x. As a result, to preserve data and ensure continued operation when you upgrade from Debezium 1.x versions to 2.3.4, you must complete some manual steps during the upgrade process.

One significant change is that the names of some connector parameters have changed. To
accommodate these changes, review the configuration properties updates in the 2023.Q2 release notes, and note the properties that are present in your connector configuration. Before you upgrade, edit the configuration of each Debezium connector to add the new names of any changed properties. Before you upgrade, edit the configuration of any 1.x connector instances so that both the old and new property names are present. After the upgrade, you can remove the old configuration options.

Prerequisites

- Debezium is now compatible with Kafka versions up to 3.5.0. This is the default Kafka version in AMQ Streams 2.5.

- The Java 11 runtime is required and must be available prior to upgrading. AMQ Streams 2.5 supports Java 11. Use Java 11 when developing new applications. Java 11 enables use of recent language updates, such as the new String API and changes in predicate support, while also benefiting from Java performance improvements. Java 8 is no longer supported in AMQ Streams 2.5.

- Check the backward-incompatible changes in the current list of breaking changes and in the 2023.Q2 release notes.

- Verify that your environment complies with the Debezium 2.3.4 Supported Configurations.

Procedure

1. From the OpenShift console, review the Kafka Connector YAML to identify the connector configuration that are no longer valid in Debezium 2.3.4. Refer to the 2023.Q2 release notes for details.

2. Edit the configuration to add the 2.x equivalents for the properties that you identify in Step 1, so that both the old and new property names are present. Set the values of the new properties to the values that were previously specified for the old properties.

3. From the OpenShift console, stop Kafka Connect to gracefully stop the connector.

4. From the OpenShift console, edit the Kafka Connect image YAML to reference the Debezium 2.3.4.Final version of the connector zip file.

5. From the OpenShift console, edit the Kafka Connector YAML to remove any configuration options that are no longer valid for your connector.

6. Adjust your application’s storage dependencies, as needed, depending on the storage module implementation dependencies in your code. For more information, see Changes to Debezium storage in the 2023.Q2 release notes.

7. Restart Kafka Connect to start the connector. After you restart the connector, it continues to process events from the point where it stopped before the upgrade. Change events records that the connector wrote to Kafka before the upgrade are not modified.

2.5. NEW FEATURES AND IMPROVEMENTS

Debezium 2.3.4 includes the following updates and improvements:

- Breaking changes

- Features promoted to General availability
2.5.1. Breaking changes

The following changes in Debezium 2.3.4 represent significant differences in connector behavior and require configuration changes that are not compatible with earlier Debezium versions: Debezium 2.3.4 introduces the following breaking changes:

- MySQL and PostgreSQL secure connection changes
- Topic and schema naming changes
- Source info block changed for Oracle connector

For information about breaking changes in the previous Debezium release, see the 2023.Q2 Release Notes.

2.5.1.1. New configuration defaults for MySQL and PostgreSQL secure connections

You can configure the Debezium connectors for MySQL and PostgreSQL to use secure SSL connections. For the MySQL connector, you specify use of a secure connection by configuring the `database.ssl.mode` property. For the PostgreSQL connector, you set the `database.sslmode` property.

Beginning with Debezium 2.3.4, these configuration options include new default values. For MySQL, the default value for `database.ssl.mode` is now `preferred`, replacing the previous default value of `disabled`. For PostgreSQL, the default value for `database.sslmode` is now `prefer`, replacing the previous default value of `disable`. Based on the new default settings, when the connectors initiate a connection to a database, they first attempt to establish an encrypted, secure connection. If a secure connection is not available, the connectors fall back to using an unsecured connection, unless configured otherwise.

2.5.1.2. Topic and schema naming changes

When Debezium generates topic names and schema names, it replaces non-ASCII characters in the names to ensure compatibility with the naming conventions of schema registries. In earlier releases, Debezium substituted an underscore character (_`) to replace non-ASCII characters. However, in some cases, after replacing non-ASCII characters, the names that Debezium generates for two topics or schema, could be identical except for their letter casing, which could lead to other problems.

In order to address this in the most compatible way, Debezium now uses a strategy-based approach to map characters uniquely. One side effect of this new approach is that Debezium no longer supports the `sanitize.field.names` configuration property. In the place of the `sanitize.field.names` property, new options are now available for specifying a naming strategy that is compatible with the conventions that you use for your tables or collections.

To specify how Debezium generates schema and field names, you can set the following properties.

- `schema.name.adjustment.mode` Specifies how schema names should be adjusted for compatibility with the message converter.
- `field.name.adjustment.mode`
Specifies how field names should be adjusted for compatibility with the message converter.

For each of the preceding properties, you can set one of the following values:

**none**
- Names are passed as-is; no adjustments are made to schema or field names.

**avro**
- Replaces characters that cannot be used in Avro with an underscore (_).

**avro_unicode**
- Replaces underscores (_) and characters that cannot be used in Avro with unicode-based escape sequences.

### 2.5.1.3. Changes to the Oracle connector source information block

The change event record that Debezium emits for insert, update, and delete event includes a payload that contain a `source` information block. For the Oracle connector, the source information block contains a special `ssn` field that represents the SQL sequence number of a change.

In some cases, the value for the `ssn` field in the source database exceeds the maximum value of an `INT32` data type (2,147,483,647). To allow for larger values, Debezium now assigns the data type `INT64` to `ssn` fields, which increases the maximum value of the field to 9,223,372,036,854,775,807.

If you currently store the `ssn` value in a sink system in your environment, or if you are using a schema registry, this change could affect the behavior of your system.

### 2.5.2. Features promoted to General Availability

The following features are promoted from Technology Preview to General Availability in the Debezium 2.3.4 release:

**Ad hoc and incremental snapshots for MongoDB connector**
- Provides a mechanism for re-running a snapshot of a table for which you previously captured a snapshot.

**Signaling for the MongoDB connector**
- Provides a mechanism for modifying the behavior of a connector, or triggering a one-time action, such as initiating an ad hoc snapshot of a table.

**Content-based routing**
- Provides a mechanism for rerouting selected events to specific topics, based on the event content.

**Filter SMT**
- Enables you to specify a subset of records that you want the connector to send to the broker.

### 2.5.3. General availability features

Debezium 2.3.4 supports the following new features:

- Automated replica identity configuration for PostgreSQL
- New notification subsystem (sink, log, JMX)
- Correlate incremental snapshot notification IDs
• Support for new signaling channels (source, Kafka, JMX)
• Oracle RAC improvements
• Oracle connector SCN-based metrics
• Server side filtering for MongoDB and Oracle
• Retry database connections on startup
• Surrogate keys for incremental snapshots
• ExtractChangedRecordState SMT
• **ExtractNewRecordState** (event flattening) SMT options for dropping fields from an event record
• **HeaderValue SMT**
• **Partition routing SMT**

2.5.3.1. Automated replica identity configuration for PostgreSQL

Debezium 2.3.4 introduces a new PostgreSQL connector feature known as "Autoset Replica Identity".

A PostgreSQL database uses replica identity to identify the columns that are captured in the database transaction logs for insert, update, and delete events. This feature enables you to configure the connector to automatically set the replica identity value for a table. When the connector starts, it reads the replica identity configuration and then sets the replica identity for the specified tables.

The new configuration property, `replica.identity.autoset.values`, specifies a comma-separated list of table and replica identity tuples. When the property specifies a replica identity for a table, that value overrides any existing replica identity configuration. For more information about PostgreSQL replica identity types, see the PostgreSQL documentation.

The `replica.identity.autoset.values` property accepts a comma-separated list of values in which each element uses the format of `<fully-qualified-table-name>:<replica-identity>`. The following example shows how to configure two tables (`table1` and `table2`) to have `FULL` replica identity:

```json
{ "replica.identity.autoset.values": "public.table1:FULL,public.table2:FULL" }
```

The user account through which the connector accesses the database requires permission to set the table’s replica identity. If the account lacks sufficient permissions, any attempt to use `replica.identity.autoset.values` results in a failure. If you cannot use the property to automatically set the replica identity, you must set the replica identity for the table manually, from a database account that has the required permission.

2.5.3.2. New notification subsystem

This release introduces a new notifications subsystem, which enables Debezium to emit events that report on the status of various connector operations, such as incremental or traditional snapshots. This new subsystem allows you to send a notification through several different channels, including Kafka topics, log files, and Java Management Extensions (JMX). These notification events can be consumed by a variety of external systems. Notification events are represented as a series of key/value tuples, including the following fields:

`id`
A UUID that identifies the notification,

**aggregate_type**

The type of notification, based on the concept of domain-driven design.

**type**

Provides more detail about the aggregate type.

**additional_data (optional)**

A map of string-based key/value pairs with additional information about the event.

The following example shows a simple notification event.

**Example notification event**

```json
{
  "id": "c485ccc3-16ff-47cc-b4e8-b56a57c3bad2",
  "aggregate_type": "Snapshot",
  "type": "Started",
  "additional_data": {
    ...
  }
}
```

In this release, Debezium supports the following types of notification events:

- Status of the initial snapshot
- Incremental snapshot progress

For more information, see Configuring notifications to report connector status.

### 2.5.3.3. Correlate incremental snapshot notification IDs

In this release, the notification and channels subsystem has been improved to correlate the signal to the notification. That is, when you send a signal and it is consumed by Debezium, the resulting notification contains an identifier that references the original signal. When communications are distributed across multiple applications and processes, this mechanism enables processes to more easily associate signals with their resulting operations.

### 2.5.3.4. Support for new signaling channels

Debezium has supported signaling since the introduction of incremental snapshots in release 1.7. Signals provide a mechanism for using metadata to instruct Debezium to perform tasks, such as writing an entries to the connector log, or performing an ad-hoc incremental snapshot.

This release introduces support for multiple signaling channels, enabling you to specify the medium that Debezium uses to watch for and react to signals. In previous versions, there was one channel supported universally across connectors, which was the database signal table. In this release, the following are available by default:

- Database signal table
- Kafka signal topic
- JMX
In this release, the signal channel subsystem has been improved to support sending signals via JMX. From a JConsole window, two subsections now exist for a connector, a notifications section, and a signal section.

The signal section enables you to invoke an operation on a JMX bean to transmit a signal to Debezium. This signal resembles the logical signal table structure in that it accepts 3 parameters:

- A unique identifier
- The signal type
- The signal payload.

For more information, see Sending signals to a Debezium connector

2.5.3.5. Oracle RAC improvements

When you use the Debezium Oracle connector with an Oracle Real Application Clusters (RAC) deployment, you must specify a rac.nodes configuration property. At minimum, the rac.nodes property must specify the host or IP address of each individual node in the cluster. Older versions of the connector also supported an alternate format in which you could specify a unique port number for each node, in recognition of the fact that different nodes might use different ports.

Debezium 2.3.4 improves Oracle RAC support by also recognizing that each node might use a different Oracle Site Identifier (SID). To account for variations in the Oracle SID configuration, you can now specify the SID parameter in the rac.nodes configuration property.

The following example illustrates connecting to two Oracle RAC nodes, each using different ports and SID parameters:

```json
{ "connector.class": "io.debezium.connector.oracle.OracleConnector", "rac.nodes": "host1.domain.com:1521/ORCLSID1,host2.domain.com:1522/ORCLSID2", ... }
```

2.5.3.6. Oracle connector SCN-based metrics

Oracle tracks a variety of system change numbers (SCNs) values in its JMX metrics, including OffsetScn, CurrentScn, OldestScn, and CommittedScn. These SCN values are numeric and can often exceed the upper bounds of a LONG data type. In past releases, Debezium exposed SCN values as String values.

To improve the utility of these metrics, Debezium now exposes these JMX metrics as BigInteger values, rather than as String values. This change enables users to view values for these metrics through tools such as Grafana and Prometheus, which do not support string-based values.

**NOTE**

If you previously gathered SCN values for other purposes, be aware they are no longer string-based, and must be interpreted as BigInteger numerical values.

2.5.3.7. Server side filtering for the MongoDB and Oracle connectors

When fetching entries from the database, the MongoDB and Oracle connectors can now submit include and exclude filters that are set in the connector configuration to the database. The MongoDB connector does this automatically. If you want the Oracle connector to submit filters when fetching entries, set the log.mining.query.filter.mode property to a value other than none, which is the default.
In past releases, the MongoDB and Oracle connectors first fetched events from the database, and then evaluated events against the filter settings. This process effectively serialized all changes from the database across the network to the connector, an approach that is inefficient, especially in high-volume environments. Connectors received some events only to discard them immediately afterwards, due to filter settings. For connectors that run in cloud environments, transmitting such a large volume of excess data inflates utilization costs.

To reduce the amount of data that the connectors fetch, in Debezium 2.3.4, connectors no longer evaluate filters after fetching data. Instead, the include and exclude lists are defined in the MongoDB change stream subscription or the Oracle fetch query. By reducing the number of events that the connector reads, the new approach results in lower network and CPU utilization. For a connector that is configured with full document or pre-image settings, this adds even more utilization to the network that is entirely unnecessary. Furthermore, by enabling the connector to receive only events that require processing, the connector is able to complete more processing, raising CPU utilization.

2.5.3.8. Retry database connections during connector startup

In previous releases, connectors used a fail-fast strategy during startup. That is, if the connector could not perform any step required to complete the startup routine, for example, connect to the database or authenticate, the connector would enter a FAILED state.

In some situations, the connector might start gracefully, run for a period of time, and then eventually encounter a fatal error. Errors could be related to resources that were not accessed during the connector’s startup lifecycle, so that you could restart the connector without encountering an error. However, when a failure results because the database becomes unavailable, if the database remains unavailable after the connector restarts, the fail-fast strategy causes the connector to enter a FAILED state. You must then intervene manually to resolve the problem.

To improve reliability and resiliency, in this release, instead of attempting to access potentially unavailable resources during startup, the connector now attempts to access these resources later in its lifecycle. In effect, during startup, Debezium is less strict about accessing potentially unavailable resources, enabling it to take advantage of the Kafka Connect retry back-off framework.

Now, if a database is unavailable during connector startup, as long as Kafka Connect retries are enabled, the connector continues to retry failed requests. A FAILED state only results after the maximum number of retry attempts has been reached, or if a non-retriable error occurs.

2.5.3.9. Use of surrogate keys in incremental snapshots

The Debezium incremental snapshot feature provides a mechanism for performing resumable, consistent snapshots of data. This ability to resume snapshots can be critical for connectors that must ingest large volumes of data.

In earlier releases, incremental snapshots required that a primary key was set for every table included in the snapshot. Beginning with Debezium 2.3.4, you can now perform incremental snapshots on key-less tables, as long as the table includes one unique that can serve as a ”surrogate key”.

Red Hat build of Debezium 2.3.4 Release Notes for Red Hat build of Debezium 2.3.4
To provide the surrogate key column data in an incremental snapshot signal, you must include the new surrogate key attribute, `surrogate-key` in the signal payload.

**An example incremental snapshot signal payload specifying a surrogate key**

```json
{
  "data-collections": [ "public.mytab" ],
  "surrogate-key": "customer_ref"
}
```

The signal in the preceding example initiates an incremental snapshot for the table `public.mytab`. The snapshot uses the `customer_ref` column as the primary key for generating snapshot windows.

**WARNING**

You must use a single column to define a surrogate key. You cannot define surrogate keys that are based on multiple columns.

You can also use the surrogate key feature with tables that have primary keys. For example, surrogate keys offer an advantage when a table’s primary key consists of multiple columns. Queries based on multiple columns generate a disjunction predicate for each column in the primary key, and the performance can be highly dependent on the environment. Using a surrogate key to reduce the number of columns in the query can provide more uniform performance.

Using a surrogate key can also provide an advantage for tables whose primary key column is based on a character-based data type. Because relational databases are generally more efficient when making numeric comparisons versus character comparisons, by specifying a numeric surrogate key, you can improve query performance.

### 2.5.3.10. **ExtractChangedRecordState SMT**

This release introduces the event record changes (**ExtractChangedRecordState**) single message transformation (SMT). You can use this transformation to identify the fields in a Debezium event record whose values changed or remained unchanged after a database operation. To use the transformation, configure it as part of your connector configuration, for example:

```yaml
transforms=changes
transforms.changes.type=io.debezium.transforms.ExtractChangedRecordState
```
transforms.changes.header.changed=ChangedFields
transforms.changes.header.unchanged=UnchangedFields

You can set the following options for this transformation to indicate different types of changes:

**header.changed**
- Shows the fields changed by an event.

**header.unchanged**
- Shows the fields that are unchanged by an event.

As in the preceding example, you can set both of these options to separately show both the changed and unchanged fields.

The transformation adds a new header with the specified name, for example, ChangedFields. It then sets the header value to a list that contains the names of the changed or unchanged fields.

For more information about using the ExtractChangedRecordState SMT, see Event record changes in the Debezium User Guide.

### 2.5.3.11. Drop event fields with new configuration options for the ExtractNewRecordState SMT

You can use the ExtractNewRecordState single message transformation (SMT) to convert Debezium change events into a simplified format for consumption by sink connectors.

This release adds three new configuration options for the transformation that you can use to drop fields from the payload or message key of an event:

**drop.fields.header.name**
- The Kafka message header name to use for listing field names in the source message that are to be dropped.

**drop.fields.from.key**
- Specifies whether to remove fields also from the key, defaults to false.

**drop.fields.keep.schema.compatible**
- Specifies whether to remove fields that are only optional, defaults to true.

**NOTE**
- To maintain schema compatibility in environments that use Avro, the SMT defaults to enforcing schema compatibility. Thus, if you configure a required field to be dropped, the SMT does not remove the field from the key or the payload, unless you disable schema compatibility.

**Emitting events that only include changed fields**

You can pair the ExtractChangedRecordState transformation with the updated ExtractNewRecordState SMT to configure a connector to emit events that only include changed fields. The following example shows a configuration that only emits changed columns in an event’s payload value:

```
transforms=changes,extract
transforms.changes.type=io.debezium.transforms.ExtractChangedRecordState
```
The preceding configuration lists unchanged fields, but it does not remove them from the event payload. If a field in the specified key did not change, it is retained, because the configuration does not explicitly change the default `false` value for `drop.fields.from.key`.

If the SMT would result in dropping a required field in the event payload, because it did not change, to comply with schema compatibility, the field is retained in the output.

For more information about the `ExtractNewRecordState` SMT, see Extracting source record after state from Debezium change events.

### 2.5.3.12. HeaderToValue SMT

Extracts specified header fields from event records, and then copies or moves the header fields to values in the event record. For more information, see Converting message headers into event record values in the Debezium User Guide.

### 2.5.3.13. Partition routing SMT

The `PartitionRouting` SMT enables you to route events to specific destination partitions based on the values of one or more specified payload fields. To calculate the destination partition, Debezium generates a hash of the specified field values.

For more information, see Routing records to partitions based on payload fields in the Debezium User Guide.

### 2.5.4. Technology Preview features

This release introduces the following Technology Preview features:

- Section 2.5.4.1, “MongoDB sharded cluster improvements (Technology Preview)"
- Section 2.5.4.2, “MongoDB incremental snapshots for multi-replica and sharded clusters (Technology Preview)”

**IMPORTANT**

Technology Preview features are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend implementing any Technology Preview features in production environments. Technology Preview features provide early access to upcoming product innovations, enabling you to test functionality and provide feedback during the development process. For more information about support scope, see Technology Preview Features Support Scope.

### 2.5.4.1. MongoDB sharded cluster improvements (Technology Preview)

In past releases, when you used the Debezium MongoDB connector in a sharded cluster deployment, the connector would open a direct connection with each replica set in a shard. This approach conflicted with the MongoDB suggestion that the connector should open a connection with the mongos router instance.
In this release aligns the connector has been redesigned to use the recommended connection strategy. If you use the connector in a sharded cluster, adjust your configuration so that the connector connects to the mongos instance. No other changes are required.

2.5.4.2. MongoDB incremental snapshots for multi-replica and sharded clusters (Technology Preview)

You can now use incremental snapshots with MongoDB multi-replica a sharded clusters. For more information, see Incremental snapshots in the MongoDB chapter of the {NameUserGuide}.

Previously available Technology Preview features

The following features that were introduced in earlier releases remain in Technology Preview:

Parallel initial snapshots

You can optionally configure SQL-based connectors to use multiple threads when performing an initial snapshot by setting the snapshot.max.threads property to a value greater than 1.

CloudEvents converter

Emits change event records that conform to the CloudEvents specification. The CloudEvents change event envelope can be JSON or Avro and each envelope type supports JSON or Avro as the data format.

Custom-developed converters

In cases where the default data type conversions do not meet your needs, you can create custom converters to use with a connector.

Use of the BLOB, CLOB, and NCLOB data types with the Oracle connector

The Oracle connector can consume Oracle large object types.

2.5.5. Developer Preview features

This Debezium 2.3.4 release includes the following Developer Preview features:

- Section 2.5.5.1, “JDBC Sink Connector (Developer Preview)”
- Section 2.5.5.2, “MySQL connector: parallel snapshots (Developer Preview)”
- Section 2.5.5.4, “Exactly-once delivery for PostgreSQL connector (Developer Preview)”
- Section 2.5.5.3, “Oracle connector: Ingesting changes from an Oracle logical standby (Developer Preview)”
## 2.5.5.1. JDBC Sink Connector (Developer Preview)

In this release Debezium introduces a JDBC sink connector implementation, breaking with a longstanding focus on developing only source connectors for relational and non-relational databases. The Debezium JDBC sink connector differs from other vendor implementations in that it is capable of ingesting raw change events emitted by Debezium connectors without first applying an event flattening transformation. The Debezium JDBC sink connector can take advantage of native Debezium source connector features, such as column type propagation, enabling you to potentially reduce the processing footprint of your data pipeline, and simplify its configuration.

The following example shows a simple configuration to ingest change events from a Kafka topic called `orders` into a PostgreSQL database. The events in the topic were emitted by a Debezium MySQL connector without using the `ExtractNewRecordState` transformation.

```json
{
   "name": "mysql-to-postgres-pipeline",
   "config": {
      "connector_class": "io.debezium.connector.jdbc.JdbcSinkConnector",
      "topics": "orders",
      "connection.url": "jdbc://postgresql://<host>:<port>/<database>",
      "connection.user": "<username>",
      "connection.password": "<password>",
      "insert.mode": "upsert",
      "delete.enabled": "true",
      "primary.key.mode": "record_key",
      "schema.evolution": "basic"
   }
}
```

The preceding example shows a series of `connection.*` properties that define the connection string and credentials for accessing a destination PostgreSQL database. When writing to the destination database, records use `UPSERT` semantics, using an insert to create a record if one doesn’t exist, or updating the record if it does. Schema evolution is enabled and a table’s key columns are derived from the event’s primary key.

You can use this release of the JDBC sink connector with the following relational databases:

- Db2
- MySQL
• Oracle
• PostgreSQL
• SQL Server

For more information, see Debezium connector for JDBC.

2.5.5.2. MySQL connector: parallel snapshots (Developer Preview)

The Debezium initial snapshot process has always been single-threaded for relational databases. This limitation primarily stems from the complexities of ensuring data consistency across multiple transactions.

Beginning in this release, you can configure a connector to use multiple threads when performing a consistent database snapshot. This implementation uses these multiple threads to execute table-level snapshots in parallel.

To take advantage parallel snapshots, set the `snapshot.max.threads` property in the connector configuration, and assign it a value greater than 1.

Example configuration using parallel snapshots

```
snapshot.max.threads=4
```

Based on the preceding example, the connector snapshots a maximum of 4 tables in parallel. If there are more tables to snapshot, after one thread finishes, the connector processes the next table in the queue. The process continues until all tables have been snapshot.

2.5.5.3. Oracle connector: Ingesting changes from an Oracle logical standby (Developer Preview)

The Debezium connector for Oracle maintains an internal `flush table` to monitor the flush cycles of the Oracle Log Writer Buffer (LGWR) process. The user account through which connector accesses the database must have permission to create and write to the flush table. Logical stand-by databases often have more restrictive rules about data manipulation and may even be read-only, therefore, writing to the database is unfavorable or even not permissible.

To enable the connector to ingest changes from an Oracle read-only logical stand-by database, this release introduces a flag that disables the creation and management of this `flush table`. You can use this Developer Preview feature with both Oracle Standalone and Oracle RAC installations.

To enable the connector to ingest changes from an Oracle read-only logical stand-by, add the following connector option:

```
internal.log.mining.read.only=true
```

2.5.5.4. Exactly-once delivery for PostgreSQL connector (Developer Preview)

Debezium has traditionally been an at-least-once delivery solution, guaranteeing that no change is ever missed. Exactly-once delivery is a proposal by the Apache Kafka community as a part of KIP-618. This proposal aims to address a common problem that producers (source connectors) encounter during a
retry. The connector might resend a batch of events to the Kafka broker even though the broker has already committed the batch. This situation can result in duplicate events being sent, which can cause problems for consumers (sink connectors) that are unable to easily handle duplicates.

No connector configuration changes are required to take advantage of exactly-once delivery. However, to enable exactly-once delivery, you must adjust your Kafka Connect worker configuration to use the configuration properties introduced in KIP-618. In Debezium 2.3.4, exactly-once semantics for PostgreSQL apply only during the streaming phase, not during snapshots.

2.5.6. Other updates in this release

This Debezium 2.3.4 release provides several feature updates and fixes, including the items in the following list:

- **DBZ-1973** Enable Debezium to send notifications about its status
- **DBZ-2296** Better control of Debezium GTID usage
- **DBZ-2979** Connector emits event records after changes to excluded columns
- **DBZ-3594** When using `snapshot.collection.include.list`, relational schema isn’t populated correctly
- **DBZ-4027** Make signalling channel configurable
- **DBZ-4488** Failed retrievable operations are retried infinitely
- **DBZ-4663** Remove option for specifying driver class from MySQL connector
- **DBZ-4829** Property `event.processing.failure.handling.mode` is not present in MySQL documentation
- **DBZ-5282** Debezium is not working with Apicurio and custom truststores
- **DBZ-5283** Add option to exclude unchanged fields in ExtractNewRecordState SMT
- **DBZ-5395** Connector offsets do not advance on transaction commit with filtered events when LOB enabled
- **DBZ-5490** Document message.key.columns and tombstone events limitations for default REPLICA IDENTITY
- **DBZ-5798** Data type conversion failed for MySQL BIGINT
- **DBZ-5917** Unable to specify column or table include list if name contains a backslash \n- **DBZ-5879** Support retrying database connection failures during connector start
- **DBZ-5907** Oracle cannot undo change
- **DBZ-5915** PostgreSQL data loss on restarts
- **DBZ-5945** Oracle multithreading lost data
- **DBZ-5966** Truncate records incompatible with ExtractNewRecordState
- **DBZ-5967** Computed partition must not be negative
DBZ-5973 MongoDB incremental snapshot not working

DBZ-5985 Table size log message for `snapshot.select.statementoverrides` tables not correct

DBZ-5988 NPE in execute snapshot signal with `exclude.tables` config on giving wrong table name

DBZ-5991 There is a problem with PostgreSQL connector parsing the boundary value of money type

DBZ-5993 Log statement for unparseable DDL statement in MySqlDatabaseSchema contains placeholder

DBZ-6001 PostgreSQL connector parses the null of the money type into 0

DBZ-6003 Nullable columns marked with "optional: false" in DDL events

DBZ-6012 PostgreSQL LSN check should honor `event.processing.failure.handling.mode`

DBZ-6026 Offsets are not flushed on connect offsets topic when encountering an error on PostgreSQL connector

DBZ-6029 Unexpected format for TIME column: 8:00

DBZ-6031 Oracle does not support compression/logging clauses after an LOB storage clause

DBZ-6037 Debezium is logging the full message along with the error

DBZ-6039 Improve resilience during internal schema history recovery from Kafka

DBZ-6046 Add Debezium steps when performing a PostgreSQL database upgrade

DBZ-6051 Incremental snapshot sends events from the signaling database to Kafka

DBZ-6064 Mask password in log statement

DBZ-6075 Loading custom offset storage fails with `Class not found` error

DBZ-6079 Increase `query.fetch.size` default to something sensible above zero

DBZ-6084 SQL Server tasks fail if the number of databases is smaller than `maxTasks`

DBZ-6089 Expose sequence field in CloudEvents message id

DBZ-6094 Reduce verbosity of skipped transactions if transaction has no events relevant to captured tables

DBZ-6107 When using LOB support, an UPDATE against multiple rows can lead to inconsistent event data

DBZ-6112 PostgreSQL: Set Replica Identity when the connector starts

DBZ-6122 PostgreSQL connector fails when processing toasted varying character arrays and date arrays

DBZ-6131 Support change stream filtering using MongoDB’s aggregation pipeline step
DBZ-6219 Highlight information about how to configure the schema history topic to store data only for intended tables

DBZ-6254 Introduce LogMiner query filtering modes

DBZ-6256 Lock contention on LOG_MINING_FLUSH table when multiple connectors deployed

DBZ-6329 The rs_id field is null in Oracle change event source information block

DBZ-6353 Using pg_replication_slot_advance which is not supported by PostgreSQL10.

DBZ-6355 log.mining.transaction.retention.hours should reference last offset and not sysdate

DBZ-6366 Code Improvements for skip.messages.without.change

DBZ-6379 Toasted hstore are not correctly processed

DBZ-6386 Oracle DDL shrink space for table partition can not be parsed

DBZ-6396 PostgreSQL connector task fails to resume streaming because replication slot is active

DBZ-6402 MongoDB connector crashes on invalid resume token

DBZ-6439 During a snapshot, the Oracle connector takes too long to read structure of captured tables

DBZ-6457 Oracle parallel snapshots do not properly set PDB context when using multitenancy

DBZ-6459 [MariaDB] Add support for userstat plugin keywords

DBZ-6474 Oracle snapshot.include.collection.list should be prefixed with databaseName in documentation.

DBZ-6485 Db2 connector can fail with NPE on notification sending

DBZ-6486 ExtractNewRecordState SMT in combination with HeaderToValue SMT results in Unexpected field name exception

DBZ-6490 BigDecimal fails when queue memory size limit is in place

DBZ-6492 Oracle table cannot be captured, got runtime.NoViableAltException

DBZ-6496 Signal poll interval has incorrect default value

DBZ-6502 Oracle JDBC driver 23.x throws ORA-18716 - not in any time zone

DBZ-6509 FileSignalChannel is not loaded

DBZ-6512 Debezium incremental snapshot chunk size documentation unclear or incorrect

DBZ-6513 Error value of negative seconds in convertOracleIntervalDaySecond

DBZ-6515 Debezium incremental snapshot chunk size documentation unclear or incorrect

DBZ-6524 [PostgreSQL] LTree data is not being captured by streaming
- **DBZ-6528** Oracle Connector: Snapshot fails with specific combination
- **DBZ-6529** Use better hashing function for PartitionRouting
- **DBZ-6533** Table order is incorrect on snapshots
- **DBZ-6543** Unhandled NullPointerException in PartitionRouting will crash the whole connect plugin
- **DBZ-6559** Bug in `field.name.adjustment.mode` property
- **DBZ-6585** Oracle unsupported DDL statement - drop multiple partitions
- **DBZ-6589** Support PostgreSQL coercion for UUID, JSON, and JSONB data types
- **DBZ-6590** MySQL parser cannot parse `CAST AS dec`
- **DBZ-6599** Oracle DDL parser does not properly detect end of statement when comments obfuscate the semicolon
- **DBZ-6605** Fixed DataCollections for table scan completion notification
- **DBZ-6610** Oracle connector is not recoverable if ORA-01327 is wrapped by another JDBC or Oracle exception
- **DBZ-6613** Fatal error when parsing MySQL (Percona 5.7.39-42) procedure
- **DBZ-6622** MySQL ALTER USER with RETAIN CURRENT PASSWORD fails with parsing exception
- **DBZ-6628** Inaccurate documentation regarding `additional-condition`
- **DBZ-6633** Oracle connection SQLRecoverableExceptions are not retried by default
- **DBZ-6643** MongoDB connector keeps going up. Fixed via DBZ-6670
- **DBZ-6648** Cannot delete non-null interval value
- **DBZ-6670** Retriable operations are retried infinitely since error handlers are not reused
- **DBZ-6677** Oracle DDL parser does not support column visibility on ALTER TABLE
- **DBZ-6690** Should use `topic.prefix` rather than `connector.server.name` in MBean namings
- **DBZ-6716** Oracle fails to process a DROP USER
- **DBZ-6724** Debezium crashes on parsing MySQL DDL statement (specific JOIN)
- **DBZ-6725** ExtractNewDocumentState for MongoDB ignore previous document state when handling delete event’s with REWRITE
- **DBZ-6733** Oracle LogMiner mining distance calculation should be skipped when upper bounds is not within distance
- **DBZ-6736** MariaDB: Unparseable DDL statement (ALTER TABLE IF EXISTS)
- **DBZ-6758** When using pgoutput in postgres connector, (+/-)Infinity is not supported in decimal values
- **DBZ-6760** Outbox transformation can cause connector to crash
- **DBZ-6774** MongoDB New Document State Extraction: nonexistent field for add.headers
- **DBZ-6777** Notifications and signals leaks between MBean instances when using JMX channels
- **DBZ-6780**Debezium crashes on parsing the MySQL DDL statement (SELECT 1;)
- **DBZ-6794**Debezium crashes on parsing the MySQL DDL statement (SELECT 1 + @sum:=1 AS ss;)
- **DBZ-6803** MySQL connector exception because the DDL parser does not accept the REPEAT function
- **DBZ-6821**Debezium crashes when DDL statements declare variable names that include non-Latin characters
- **DBZ-6824** When parsing MySQL DDL, the connector now properly trims default values for the BIGINT and SMALLINT types
- **DBZ-6830** Partial and multi-response transactions are now logged in debug mode only
- **DBZ-6867** Streaming aggregation pipeline broken for combination of database filter and signal collection

### 2.6. DEPRECATED FEATURES

The following features are deprecated in this release:

The **mongodb.hosts** property is no longer supported. To configure Debezium connector to connect to a MongoDB replica set, use the **mongodb.connection.string** property.

### 2.7. KNOWN ISSUES

The following known issue affects Debezium 2.3.4:

- Apicurio registry 2.4.3 and 2.4.4 causes endless rebalance loop on Kafka