Red Hat Integration 2023.q4

Installing Debezium on RHEL

For use with Red Hat Integration 2.3.4 on Red Hat Enterprise Linux (RHEL)

Last Updated: 2023-11-17

Red Hat Integration 2023.q4 Installing Debezium on RHEL

For use with Red Hat Integration 2.3.4 on Red Hat Enterprise Linux (RHEL)

Legal Notice

Abstract

This guide describes how to install Red Hat Integration on RHEL with AMQ Streams.

Table of Contents

PREFACE

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation.

To propose improvements, open a Jira issue and describe your suggested changes. Provide as much detail as possible to enable us to address your request quickly.

Prerequisite

- You have a Red Hat Customer Portal account. This account enables you to log in to the Red Hat Jira Software instance.
  If you do not have an account, you will be prompted to create one.

Procedure

1. Click the following link: Create issue.
2. In the **Summary** text box, enter a brief description of the issue.
3. In the **Description** text box, provide the following information:
   - The URL of the page where you found the issue.
   - A detailed description of the issue.
     You can leave the information in any other fields at their default values.
4. Click **Create** to submit the Jira issue to the documentation team.

Thank you for taking the time to provide feedback.

CHAPTER 1. DEBEZIUM OVERVIEW

Debezium for Red Hat Integration is a distributed platform that captures database operations, creates data change event records for row-level operations, and streams change event records to Apache Kafka topics. Debezium is built on Apache Kafka and is deployed and integrated with AMQ Streams.

Debezium captures row-level changes to a database table and passes corresponding change events to AMQ Streams. Applications can read these *change event streams* and access the change events in the order in which they occurred.

Debezium is the upstream community project for Debezium for Red Hat Integration.

Debezium has multiple uses, including:

- Data replication
- Updating caches and search indexes
- Simplifying monolithic applications
- Data integration
- Enabling streaming queries

Debezium provides Apache Kafka Connect connectors for the following common databases:

- Db2
- MySQL
- MongoDB
- Oracle
- PostgreSQL
- SQL Server

CHAPTER 2. INSTALLING DEBEZIUM CONNECTORS ON RHEL

Install Debezium connectors through AMQ Streams by extending Kafka Connect with connector plugins. Following a deployment of AMQ Streams, you can deploy Debezium as a connector configuration through Kafka Connect.

2.1. KAFKA TOPIC CREATION RECOMMENDATIONS

Debezium stores data in multiple Apache Kafka topics. The topics must either be created in advance by an administrator, or you can configure Kafka Connect to configure topics automatically.

The following list describes limitations and recommendations to consider when creating topics:

Database schema history topics for the Debezium Db2, MySQL, Oracle, and SQL Server connectors

For each of the preceding connectors a database schema history topic is required. Whether you manually create the database schema history topic, use the Kafka broker to create the topic automatically, or use Kafka Connect to create the topic , ensure that the topic is configured with the following settings:

- Infinite or very long retention.
- Replication factor of at least three in production environments.
- Single partition.

Other topics

- When you enable Kafka log compaction so that only the *last* change event for a given record is saved, set the following topic properties in Apache Kafka:
  - **min.compaction.lag.ms**
  - **delete.retention.ms**
    To ensure that topic consumers have enough time to receive all events and delete markers, specify values for the preceding properties that are larger than the maximum downtime that you expect for your sink connectors. For example, consider the downtime that might occur when you apply updates to sink connectors.
- Replicated in production.
- Single partition.
  You can relax the single partition rule, but your application must handle out-of-order events for different rows in the database. Events for a single row are still totally ordered. If you use multiple partitions, the default behavior is that Kafka determines the partition by hashing the key. Other partition strategies require the use of single message transformations (SMTs) to set the partition number for each record.

2.2. PLANNING THE DEBEZIUM CONNECTOR CONFIGURATION

Before you deploy a Debezium connector, determine how you want to configure the connector. The configuration provides information that specifies the behavior of the connector and enables Debezium to connect to the source database.

You specify the connector configuration as JSON, and when you are ready to register the connector, you use **curl** to submit the configuration to the Kafka Connect API endpoint.

Prerequisites

- A source database is deployed and the Debezium connector can access the database.
- You know the following information, which the connector requires to access the source database:
  - Name or IP address of the database host.
  - Port number for connecting to the database.
  - Name of the account that the connector can use to sign in to the database.
  - Password of the database user account.
  - Name of the database.
  - The names of the tables from which you want the connector to capture information.
  - The name of the Kafka broker to which you want the connector to emit change events.
  - The name of the Kafka topic to which you want the connector to send database history information.

Procedure

- Specify the configuration that you want to apply to the Debezium connector in JSON format. The following example shows a simple configuration for a Debezium MySQL connector:

```
{
 "name": "inventory-connector", 1
 "config": {
  "connector.class": "io.debezium.connector.mysql.MySqlConnector", 2
  "tasks.max": "1", 3
  "database.hostname": "mysql", 4
  "database.port": "3306", 5
  "database.user": "debezium", 6
  "database.password": "dbz", 7
  "database.server.id": "184054", 8
  "topic.prefix": "dbserver1", 9
  "table.include.list": "public.inventory", 10
  "schema.history.internal.kafka.bootstrap.servers": "kafka:9092", 11
  "schema.history.internal.kafka.topic": "dbhistory.inventory" 12
 }
}
```

1. The name of the connector to register with the Kafka Connect cluster.
2. The name of the connector class.
3. The number of tasks that can operate concurrently. Only one task should operate at a time.

4. The host name or IP address of the host database instance.
5. The port number of the database instance.
6. The name of the user account through which Debezium connects to the database.
7. The password for the database user account.
8. A unique numeric ID for the connector.
   This property is used for the MySQL connector only.
9. A string that serves as the logical identifier for the database server or cluster of servers from which the connector captures changes. The specified string designates a namespace. Debezium prefixes this name to each Kafka topic that the connector writes to, as well as to the names of Kafka Connect schemas, and the namespaces of the corresponding Avro schema, when the Avro converter is used.
10. The list of tables from which the connector captures change events.
11. The name of the Kafka broker where the connector sends the database schema history. The specified broker also receives the change events that the connector emits.
12. The name of the Kafka topic that stores the schema history.
    After a connector restart, the connector resumes reading the database log from the point at which it stopped, emitting events for any transactions that occurred while it was offline. Before the connector writes change events for an unread transaction to Kafka, it checks the schema history and then applies the schema that was in effect when the original transaction occurred.

Additional information

- For information about the configuration properties that you can set for each type of connector, see the deployment documentation for the connector in the Debezium User Guide.

Next steps

Section 2.3, “Deploying Debezium with AMQ Streams on Red Hat Enterprise Linux” .

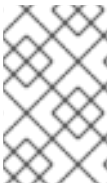
2.3. DEPLOYING DEBEZIUM WITH AMQ STREAMS ON RED HAT ENTERPRISE LINUX

This procedure describes how to set up connectors for Debezium on Red Hat Enterprise Linux. Connectors are deployed to an AMQ Streams cluster using Apache Kafka Connect, a framework for streaming data between Apache Kafka and external systems. Kafka Connect must be run in distributed mode rather than standalone mode.

Prerequisites

- The host environment to which you want to deploy Debezium runs Red Hat Enterprise Linux, AMQ Streams, and Java in a supported configuration.
  - For information about how to install AMQ Streams, see Installing AMQ Streams.

- For information about how to install a basic, non-production AMQ Streams cluster that contains a single ZooKeeper node, and a single Kafka node, see Running a single node AMQ Streams cluster.



NOTE

If you are running an earlier version of AMQ Streams, you must first upgrade to AMQ Streams 2.5. For information about the upgrade process, see AMQ Streams and Kafka upgrades.

- You have administrative privileges (**sudo** access) on the host.
- Apache ZooKeeper and the Apache Kafka broker are running.
- Kafka Connect is running in distributed mode, and not in standalone mode.
- You know the credentials of the **kafka** user that was created when AMQ Streams was installed.
- A source database is deployed and the host where you deploy Debezium has access to the database.
- You know how you want to configure the connector.

Procedure

1. Download the Debezium connector or connectors that you want to use from the Red Hat Integration download site. For example, to use Debezium with a MySQL database, download the **Debezium 2.3.4 MySQL Connector**.
2. On the Red Hat Enterprise Linux host where you deployed AMQ Streams, open a terminal window and create a **connector-plugins** directory in **/opt/kafka**, if it does not already exist:

```
$ sudo mkdir /opt/kafka/connector-plugins
```

3. Enter the following command to extract the contents of the Debezium connector archive that you downloaded to the **/opt/kafka/connector-plugins** directory.

```
$ sudo unzip debezium-connector-mysql-2.3.4.Final.zip -d /opt/kafka/connector-plugins
```

4. Repeat Steps 1 -3 for each connector that you want to install.
5. From a terminal window, sign in as the **kafka** user:

```
$ su - kafka
$ Password:
```

6. Stop the Kafka Connect process if it is running.
   a. Check whether Kafka Connect is running in distributed mode by entering the following command:

```
$ jcmd | grep ConnectDistributed
```

   If the process is running, the command returns the process ID, for example:

```
18514 org.apache.kafka.connect.cli.ConnectDistributed /opt/kafka/config/connect-distributed.properties
```

b. Stop the process by entering the **kill** command with the process ID, for example,

```
$ kill 18514
```

7. Edit the **connect-distributed.properties** file in **/opt/kafka/config/** and set the value of **plugin.path** to the location of the parent directory for the Debezium connector plug-ins:

```
plugin.path=/opt/kafka/connector-plugins
```

8. Start Kafka Connect in distributed mode.

```
$ /opt/kafka/bin/connect-distributed.sh /opt/kafka/config/connect-distributed.properties
```

9. After Kafka Connect is running, use the Kafka Connect API to register the connector.
Enter a **curl** command to submit a **POST** request that sends the connector configuration JSON that you specified in Section 2.2, "Planning the Debezium connector configuration" to the Kafka Connect REST API endpoint at **localhost:8083/connectors**.
For example:

```
curl -i -X POST -H "Accept:application/json" -H "Content-Type:application/json" localhost:8083/connectors/ \
-d '{"name": "inventory-connector", "config": \
{ "connector.class": "io.debezium.connector.mysql.MySqlConnector", \
"tasks.max": "1", \
"database.hostname": "mysql", \
"database.port": "3306", \
"database.user": "debezium", \
"database.password": "dbz", \
"database.server.id": "184054", \
"topic.prefix": "dbserver1", \
"table.include.list": "public.inventory", \
"schema.history.internal.kafka.bootstrap.servers": "kafka:9092", \
"schema.history.internal.kafka.topic": "dbhistory.inventory" } }'
```

To register multiple connectors, submit a separate request for each one.

10. Restart Kafka Connect to implement your changes.
As Kafka Connect starts, it loads the configured Debezium connectors from the **connector-plugins** directory.

After you complete the configuration, the deployed connector connects to the source database and produces events for each inserted, updated, or deleted row or document.

11. Repeat Steps 5–10 for each Kafka Connect worker node.

Next steps

Verify the deployment.

Additional resources

- Adding connector plugins

2.4. VERIFYING THE DEPLOYMENT

After the connector starts, it performs a snapshot of the configured database, and creates topics for each table that you specify.

Prerequisites

- You deployed a connector on Red Hat Enterprise Linux, based on the instructions in Section 2.3, “Deploying Debezium with AMQ Streams on Red Hat Enterprise Linux”. .Procedure

1. From a terminal window on the host, enter the following command to request the list of connectors from the Kafka Connect API:

```
$ curl -H "Accept:application/json" localhost:8083/connectors/
```

   The query returns the name of the deployed connector, for example:

```
["inventory-connector"]
```

2. From a terminal window on the host, enter the following command to view the tasks that the connector is running:

```
$ curl -i -X GET -H "Accept:application/json" localhost:8083/connectors/inventory-connector
```

   The command returns output that is similar to the following example:

```
HTTP/1.1 200 OK
Date: Thu, 06 Feb 2020 22:12:03 GMT
Content-Type: application/json
Content-Length: 531
Server: Jetty(9.4.20.v20190813)

{
  "name": "inventory-connector",
  ...
  "tasks": [
    {
      "connector": "inventory-connector",
      "task": 0
    }
  ]
}
```

3. Display a list of topics in the Kafka cluster.
   From a terminal window, navigate to **/opt/kafka/bin/** and run the following shell script:

```
./kafka-topics.sh --bootstrap-server=localhost:9092 --list
```

   The Kafka broker returns a list of topics that the connector creates. The available topics depends on the settings of the connector’s **snapshot.mode**,**snapshot.include.collection.list**, and **table.include.list** configuration

properties. By default, the connector creates a topic for each non-system table in the database.

4. View the contents of a topic.
   From a terminal window, navigate to **/opt/kafka/bin/**, and run the **kafka-console-consumer.sh** shell script to display the contents of one of the topics returned by the preceding command:
   For example:

```
./kafka-console-consumer.sh \
>     --bootstrap-server localhost:9092 \
>     --from-beginning \
>     --property print.key=true \
>     --topic=dbserver1.inventory.products_on_hand
```

For each event in the topic, the command returns information that is similar to the following output:

Example 2.1. Content of a Integration change event


```
{"schema":{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"}],"optional":false,"name":"dbserver1.inventory.products_on_hand.Key"},"payload":{"product_id":101}} {"schema":
{"type":"struct","fields":[{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"dbserver1.inventory.products_on_hand.Value","field":"before"},{"type":"struct","fields":
[{"type":"int32","optional":false,"field":"product_id"},
{"type":"int32","optional":false,"field":"quantity"}],"optional":true,"name":"dbserver1.inventory.products_on_hand.Value","field":"after"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"version"},
{"type":"string","optional":false,"field":"connector"},
{"type":"string","optional":false,"field":"name"},
{"type":"int64","optional":false,"field":"ts_ms"},
{"type":"string","optional":true,"name":"io.debezium.data.Enum","version":1,"parameters":{"allowed":"true,last,false"},"default":"false","field":"snapshot"},
{"type":"string","optional":false,"field":"db"},
{"type":"string","optional":true,"field":"sequence"},
{"type":"string","optional":true,"field":"table"},
{"type":"int64","optional":false,"field":"server_id"},
{"type":"string","optional":true,"field":"gtid"},{"type":"string","optional":false,"field":"file"},
{"type":"int64","optional":false,"field":"pos"},
{"type":"int32","optional":false,"field":"row"},
{"type":"int64","optional":true,"field":"thread"},
{"type":"string","optional":true,"field":"query"}],"optional":false,"name":"io.debezium.connector.mysql.Source","field":"source"},{"type":"string","optional":false,"field":"op"},
{"type":"int64","optional":true,"field":"ts_ms"},{"type":"struct","fields":
[{"type":"string","optional":false,"field":"id"},
{"type":"int64","optional":false,"field":"total_order"},
{"type":"int64","optional":false,"field":"data_collection_order"}],"optional":true,"field":"transaction"}],"optional":false,"name":"dbserver1.inventory.products_on_hand.Envelope"},
"payload":{"before":null,"after":{"product_id":101,"quantity":3},"source":
{"version":"2.3.4.Final-redhat-
00001","connector":"mysql","name":"inventory_connector_mysql","ts_ms":1638985247805,"snapshot":"true","db":"inventory","sequence":null,"table":"products_on_hand","serve
```

```
r_id":0,"gtid":null,"file":"mysql-
bin.000003","pos":156,"row":0,"thread":null,"query":null},"op":"r","ts_ms":16389852478
05,"transaction":null}}
```

In the preceding example, the **payload** value shows that the connector snapshot generated a read (**"op" ="r"**) event from the table **inventory.products_on_hand**. The **"before"** state of the **product_id** record is **null**, indicating that no previous value exists for the record. The **"after"** state shows a **quantity** of **3** for the item with **product_id 101**.

Next Steps

For information about the configuration settings that are available for each connector, and to learn how to configure source databases to enable change data capture, see the Debezium User Guide.

2.5. UPDATING DEBEZIUM CONNECTOR PLUG-INS IN THE KAFKA CONNECT CLUSTER

To replace the version of a Debezium connector that is deployed on Red Hat Enterprise Linux, you update the connector plug-in.

Procedure

1. Download a copy of the Debezium connector plug-in that you want to replace from the Red Hat Integration download site.
2. Extract the contents of the Debezium connector archive to the **/opt/kafka/connector-plugins** directory.

```
$ sudo unzip debezium-connector-mysql-2.3.4.Final.zip -d /opt/kafka/connector-plugins
```

3. Restart Kafka Connect.

APPENDIX A. USING YOUR SUBSCRIPTION

Integration is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

Accessing your account

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

Activating a subscription

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

Downloading zip and tar files

To access zip or tar files, use the customer portal to find the relevant files for download. If you are using RPM packages, this step is not required.

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.
2. Scroll down to **INTEGRATION AND AUTOMATION**.
3. Click **Red Hat Integration** to display the Red Hat Integration downloads page.
4. Click the **Download** link for your component.

Revised on 2023-11-17 04:11:22 UTC