debezium

Debezium Server

Table of Contents

NOTE

NOTE

Please let us know if you encounter any problems while using this feature. Also please reach out if you have requirements for specific sinks to be supported by Debezium Server or even would be interested in contributing the required implementation.

Debezium provides a ready-to-use application that streams change events from a source database to messaging infrastructure like Amazon Kinesis, Google Cloud Pub/Sub, Apache Pulsar, Redis (Stream), or NATS JetStream. For streaming change events to Apache Kafka, it is recommended to deploy the Debezium connectors via Kafka Connect.

Installation

To install the server download and unpack the server distribution archive:

- Debezium Server distribution

A directory named `debezium-server` will be created with these contents:

```
debezium-server/
|-- CHANGELOG.md
|-- conf
|-- CONTRIBUTE.md
|-- COPYRIGHT.txt
|-- debezium-server-2.7.1.Final-runner.jar
|-- lib
|-- LICENSE-3rd-PARTIES.txt
|-- LICENSE.txt
|-- README.md
`-- run.sh
```

The server is started using `run.sh` script, dependencies are stored in the `lib` directory, and the directory `conf` contains configuration files.

NOTE

In case of using the Oracle connector you will have to add to the `lib` directory the ORACLE JDBC driver (if using XStream also the XStream API files), explained here: Obtaining the Oracle JDBC driver and XStream API files

Configuration

Debezium Server uses MicroProfile Configuration for configuration. This means that the application can be configured from disparate sources like configuration files, environment variables, system properties etc.

The main configuration file is *conf/application.properties*. There are multiple sections configured:

- `debezium.source` is for source connector configuration; each instance of Debezium Server runs exactly one connector
- `debezium.sink` is for the sink system configuration
- `debezium.format` is for the output serialization format configuration
- `debezium.transforms` is for the configuration of message transformations
- `debezium.predicates` is for the configuration of message transformation predicates

An example configuration file can look like so:

```
debezium.sink.type=kinesis
debezium.sink.kinesis.region=eu-central-1
debezium.source.connector.class=io.debezium.connector.postgresql.PostgresConnector
debezium.source.offset.storage.file.filename=data/offsets.dat
debezium.source.offset.flush.interval.ms=0
debezium.source.database.hostname=localhost
debezium.source.database.port=5432
debezium.source.database.user=postgres
debezium.source.database.password=postgres
debezium.source.database.dbname=postgres
debezium.source.topic.prefix=tutorial
debezium.source.schema.include.list=inventory
```

In this configuration file example:

- The sink is setup for AWS Kinesis in region `eu-central-1`
- The source connector is setup for PostgreSQL using the default Debezium decoderbufs plugin. If using PostgreSQL's built-in `pgoutput` plugin, set `debezium.source.plugin.name=pgoutput`
- The source connector is set to capture events from a schema named `inventory`. If you want to capture all changes in the database, remove this line. Otherwise, update this line to correspond to your preferred schema or tables.
- The source offset will be stored in a file named `offsets.dat` in the `data` directory. Note that you might need to create this directory to prevent an error on startup.

When the server is started it generates a seqeunce of log messages like this:

```
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2020-05-15 11:33:12,189 INFO  [io.deb.ser.kin.KinesisChangeConsumer] (main)
Using
'io.debezium.server.kinesis.KinesisChangeConsumer$$Lambda$119/0x0000000840130c40@f
stream name mapper
```

```
2020-05-15 11:33:12,628 INFO  [io.deb.ser.kin.KinesisChangeConsumer] (main)
Using default KinesisClient
'software.amazon.awssdk.services.kinesis.DefaultKinesisClient@d1f74b8'
2020-05-15 11:33:12,628 INFO  [io.deb.ser.DebeziumServer] (main) Consumer
'io.debezium.server.kinesis.KinesisChangeConsumer' instantiated
2020-05-15 11:33:12,754 INFO  [org.apa.kaf.con.jso.JsonConverterConfig] (main)
JsonConverterConfig values:
        converter.type = key
        decimal.format = BASE64
        schemas.cache.size = 1000
        schemas.enable = true

2020-05-15 11:33:12,757 INFO  [org.apa.kaf.con.jso.JsonConverterConfig] (main)
JsonConverterConfig values:
        converter.type = value
        decimal.format = BASE64
        schemas.cache.size = 1000
        schemas.enable = false

2020-05-15 11:33:12,763 INFO  [io.deb.emb.EmbeddedEngine$EmbeddedConfig] (main)
EmbeddedConfig values:
        access.control.allow.methods =
        access.control.allow.origin =
        admin.listeners = null
        bootstrap.servers = [localhost:9092]
        client.dns.lookup = default
        config.providers = []
        connector.client.config.override.policy = None
        header.converter = class
org.apache.kafka.connect.storage.SimpleHeaderConverter
        internal.key.converter = class
org.apache.kafka.connect.json.JsonConverter
        internal.value.converter = class
org.apache.kafka.connect.json.JsonConverter
        key.converter = class org.apache.kafka.connect.json.JsonConverter
        listeners = null
        metric.reporters = []
        metrics.num.samples = 2
        metrics.recording.level = INFO
        metrics.sample.window.ms = 30000
        offset.flush.interval.ms = 0
        offset.flush.timeout.ms = 5000
        offset.storage.file.filename = data/offsets.dat
        offset.storage.partitions = null
        offset.storage.replication.factor = null
        offset.storage.topic =
        plugin.path = null
        rest.advertised.host.name = null
        rest.advertised.listener = null
        rest.advertised.port = null
        rest.extension.classes = []
```

```
        rest.host.name = null
        rest.port = 8083
        ssl.client.auth = none
        task.shutdown.graceful.timeout.ms = 5000
        topic.tracking.allow.reset = true
        topic.tracking.enable = true
        value.converter = class org.apache.kafka.connect.json.JsonConverter

2020-05-15 11:33:12,763 INFO  [org.apa.kaf.con.run.WorkerConfig] (main) Worker
configuration property 'internal.key.converter' is deprecated and may be removed
in an upcoming release. The specified value
'org.apache.kafka.connect.json.JsonConverter' matches the default, so this
property can be safely removed from the worker configuration.
2020-05-15 11:33:12,763 INFO  [org.apa.kaf.con.run.WorkerConfig] (main) Worker
configuration property 'internal.value.converter' is deprecated and may be
removed in an upcoming release. The specified value
'org.apache.kafka.connect.json.JsonConverter' matches the default, so this
property can be safely removed from the worker configuration.
2020-05-15 11:33:12,765 INFO  [org.apa.kaf.con.jso.JsonConverterConfig] (main)
JsonConverterConfig values:
        converter.type = key
        decimal.format = BASE64
        schemas.cache.size = 1000
        schemas.enable = true

2020-05-15 11:33:12,765 INFO  [org.apa.kaf.con.jso.JsonConverterConfig] (main)
JsonConverterConfig values:
        converter.type = value
        decimal.format = BASE64
        schemas.cache.size = 1000
        schemas.enable = true

2020-05-15 11:33:12,767 INFO  [io.deb.ser.DebeziumServer] (main) Engine executor
started
2020-05-15 11:33:12,773 INFO  [org.apa.kaf.con.sto.FileOffsetBackingStore]
(pool-3-thread-1) Starting FileOffsetBackingStore with file data/offsets.dat
2020-05-15 11:33:12,835 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
Starting PostgresConnectorTask with configuration:
2020-05-15 11:33:12,837 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
connector.class = io.debezium.connector.postgresql.PostgresConnector
2020-05-15 11:33:12,837 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
offset.flush.interval.ms = 0
2020-05-15 11:33:12,838 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
database.user = postgres
2020-05-15 11:33:12,838 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
database.dbname = postgres
2020-05-15 11:33:12,838 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
offset.storage.file.filename = data/offsets.dat
2020-05-15 11:33:12,838 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
database.hostname = localhost
2020-05-15 11:33:12,838 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
```

```
database.password = ********
2020-05-15 11:33:12,839 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
name = kinesis
2020-05-15 11:33:12,839 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
topic.prefix = tutorial
2020-05-15 11:33:12,839 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
database.port = 5432
2020-05-15 11:33:12,839 INFO  [io.deb.con.com.BaseSourceTask] (pool-3-thread-1)
schema.include.list = inventory
2020-05-15 11:33:12,908 INFO  [io.quarkus] (main) debezium-server 1.2.0-SNAPSHOT
(powered by Quarkus 1.4.1.Final) started in 1.198s. Listening on:
http://0.0.0.0:8080
2020-05-15 11:33:12,911 INFO  [io.quarkus] (main) Profile prod activated.
2020-05-15 11:33:12,911 INFO  [io.quarkus] (main) Installed features: [cdi,
smallrye-health]
```

Source configuration

The source configuration uses the same configuration properties that are described on the specific connector documentation pages (just with `debezium.source` prefix), together with few more specific ones, necessary for running outside of Kafka Connect:

Property	Default
debezium.source.connector.class	
debezium.source.offset.storage	org.apache.kafk
debezium.source.offset.storage.file.filename	
debezium.source.offset.flush.interval.ms	
debezium.source.offset.storage.redis.address	

Property	Default
debezium.source.offset.storage.redis.user	
debezium.source.offset.storage.redis.password	
debezium.source.offset.storage.redis.ssl.enabled	
debezium.source.offset.storage.redis.key	
debezium.source.offset.storage.redis.wait.enabled	false
debezium.source.offset.storage.redis.wait.timeout.ms	1000
debezium.source.offset.storage.redis.wait.retry.enabled	false
debezium.source.offset.storage.redis.wait.retry.delay.ms	1000
debezium.source.schema.history.internal	io.debezium.sto
debezium.source.schema.history.internal.file.filename	
debezium.source.schema.history.internal.redis.address	

Property	Default
debezium.source.schema.history.internal.redis.user	
debezium.source.schema.history.internal.redis.password	
debezium.source.schema.history.internal.redis.ssl.enabled	
debezium.source.schema.history.internal.redis.key	
debezium.source.schema.history.internal.redis.retry.initial.delay.ms	
debezium.source.schema.history.internal.redis.retry.max.delay.ms	
debezium.source.schema.history.internal.redis.retry.max.attempts	
debezium.source.schema.history.internal.redis.connection.timeout.ms	
debezium.source.schema.history.internal.redis.socket.timeout.ms	
debezium.source.schema.history.internal.redis.wait.enabled	false
debezium.source.schema.history.internal.redis.wait.timeout.ms	1000
debezium.source.schema.history.internal.redis.wait.retry.enabled	false
debezium.source.schema.history.internal.redis.wait.retry.delay.ms	1000
debezium.source.schema.history.internal.rocketmq.topic	
debezium.source.schema.history.internal.rocketmq.name.srv.addr	localhost:9876
debezium.source.schema.history.internal.rocketmq.acl.enabled	false
debezium.source.schema.history.internal.rocketmq.access.key	
debezium.source.schema.history.internal.rocketmq.secret.key	
debezium.source.schema.history.internal.rocketmq.recovery.attempts	60

Property	Default
debezium.source.schema.history.internal.rocketmq.recovery.poll.interval.ms	1000
debezium.source.schema.history.internal.rocketmq.store.record.timeout.ms	60000

Format configuration

The message output format can be configured for both key and value separately. By default the output is in JSON format but an arbitrary implementation of Kafka Connect's `Converter` can be used.

Property	Default	Description
debezium.format.key	json	The name of the output format for key, one of json / jsonbytearray / avro / protobuf / simplestring / binary.
debezium.format.key.*		Configuration properties passed to the key converter.
debezium.format.value	json	The name of the output format for value, one of json / jsonbytearray / avro / protobuf / cloudevents / simp nary.
debezium.format.value.*		Configuration properties passed to the value converter.
debezium.format.header	json	The name of the output format for value, one of json / jsonbytearray.
debezium.format.header.*		Configuration properties passed to the header converter.

Transformation configuration

Before the messages are delivered to the sink, they can run through a sequence of transformations. The server supports single message transformations defined by Kafka Connect. The configuration will need to contain the list of transformations, implementation class for each transformation and configuration options for each of the transformations.

Property	Default	Description [id="debezium-transforms"]
debezium.transforms		The comma separated list of symbolic names of transformations.
debezium.transforms.<name>.type		The name of Java class implementing the transformation with name <name>.

Property	Default	Description [id="debezium-transforms"]
`debezium.transforms.<name>.*`		Configuration properties passed to the transformation with name `<name>`.
`debezium.transforms.<name>.predicate`		The name of the predicate to be applied to the transformation with name `<name>`.
`debezium.transforms.<name>.negate`	false	Determines if the result of the predicate to the transformation with name `<name>` will be negated.

Predicates configuration

A Predicate can be associated with a transformation in order to make the transformation optional. The server supports Filter and Conditional SMTs defined by Kafka Connect. The configuration will need to contain the list of predicates, implementation class for each predicate and configuration options for each of the predicates.

Property	Default	Description [id="debezium-predicates"]
`debezium.predicates`		The comma separated list of symbolic names of predicates.
`debezium.predicates.<name>.type`		The name of Java class implementing the predicate with name `<name>`.
`debezium.predicates.<name>.*`		Configuration properties passed to the predicate with name `<name>`.

Additional configuration

Debezium Server runs on top of the Quarkus framework. All configuration options exposed by Quarkus are available in Debezium Server too. The most frequent used are:

Property	Default	Description [id="debezium-quarkus-http-port"]
`quarkus.http.port`	8080	The port on which Debezium exposes Microprofile Health endpoint and other exposed status information. Health can be accessed on `http://host:8080/q/health`.
`quarkus.log.level`	INFO	The default log level for every log category.
`quarkus.log.console.json`	true	Determine whether to enable the JSON console formatting extension, which disables "normal" console formatting.

JSON logging can be disabled by setting `quarkus.log.console.json=false` in the *conf/application.properties* file, as demonstrated in the *conf/application.properties.example* file.

Enabling message filtering

Debezium Server provides filter STM capability, see Message Filtering for more details. However, for security reasons it's not enabled by default and has to be explicitly enabled when Debezium Server is started. To enable it, set environment variable `ENABLE_DEBEZIUM_SCRIPTING` to `true`. This will add `debezium-scripting` jar file and JSR 223 implementations (currently Groovy and graalvm.js) jar files into the server class path. These jar files are contained in `opt_lib` directory of the Debezium Server distribution.

Sink configuration

Sink configuration is specific for each sink type.

The sink is selected by configuration property `debezium.sink.type`.

Amazon Kinesis

Amazon Kinesis is an implementation of data streaming system with support for stream sharding and other techniques for high scalability. Kinesis exposes a set of REST APIs and provides a (not-only) Java SDK that is used to implement the sink.

Property	Default	Description
`debezium.sink.type`		Must be set to `kinesis`.
`debezium.sink.kinesis.region`		A region name in which the Kinesis target streams are provided.
`debezium.sink.kinesis.endpoint`	*endpoint determined by aws sdk*	(Optional) An endpoint url at which the Kinesis target streams are provided.
`debezium.sink.kinesis.credentials.profile`		(Optional) A credentials profile name used to communicate with Amazon API through the default credential profiles file. If not present will be used the default credentials provider chain. It will look for credentials on the following order: environment variables, java system properties, web identity token credentials, default credential profiles file, Amazon ECS container credentials and instance profile credentials.
`debezium.sink.kinesis.null.key`	`default`	Kinesis does not support the notion of messages without key. So this string will be used as message key for messages from tables without primary key.

Injection points

The Kinesis sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
`software.amazon.awssdk.services.kinesis.KinesisClient`	`@CustomConsumerBuilder`	Custom config- ured instance c a `KinesisClie` used to send messages to ta get streams.
`io.debezium.server.StreamNameMapper`		Custom imple- mentation map the planned de tination (topic) name into a physical Kinesi stream name. I default the sam name is used.

Google Cloud Pub/Sub

Google Cloud Pub/Sub is an implementation of messaging/eventing system designed for scalable batch and stream processing applications. Pub/Sub exposes a set of REST APIs and provides a (not-only) Java SDK that is used to implement the sink.

Property	Default	Description
`debezium.sink.type`		Must be set to `pubsub` .
`debezium.sink.pubsub.project.id`	*system-wide de- fault project id*	A project name in which the target topics are created.

Property	Default	Description
debezium.sink.pubsub.ordering.enabled	true	Pub/Sub can optionally use a message key to guarantee the delivery of the messages in the same order as were sent for messages with the same order key. This feature can be disabled.
debezium.sink.pubsub.null.key	default	Tables without primary key sends messages with null key. This is not supported by Pub/Sub so a surrogate key must be used.
debezium.sink.pubsub.batch.delay.threshold.ms	100	The maximum amount of time to wait to reach element count or request bytes threshold before publishing outstanding messages to Pub/Sub.
debezium.sink.pubsub.batch.element.count.threshold	100L	Once this many messages are queued, send all of the messages in a single call, even if the delay threshold hasn't elapsed yet.

Property	Default	Description
debezium.sink.pubsub.batch.request.byte.threshold	10000000L	Once the number of bytes in the batched request reaches this threshold, send all of the messages in a single call, even if neither the delay or message count thresholds have been exceeded yet.
debezium.sink.pubsub.flowcontrol.enabled	false	When enabled, configures your publisher client with flow control to limit the rate of publish requests.
debezium.sink.pubsub.flowcontrol.max.outstanding.messages	Long.MAX_VALUE	(Optional) If flow control enabled, the maxmium number of messages before messages are blocked from being published
debezium.sink.pubsub.flowcontrol.max.outstanding.bytes	Long.MAX_VALUE	(Optional) If flow control enabled, the maxmium number of bytes before messages are blocked from being published
debezium.sink.pubsub.retry.total.timeout.ms	60000	The total timeout for a call to publish (including retries) to Pub/Sub.
debezium.sink.pubsub.retry.initial.delay.ms	5	The initial amount of time to wait before retrying the request.

Property	Default	Description
debezium.sink.pubsub.retry.delay.multiplier	2.0	The previous wait time is multiplied by this multiplier to come up with the next wait time, until the max is reached.
debezium.sink.pubsub.retry.max.delay.ms	Long.MAX_VALUE	The maximum amount of time to wait before retrying. i.e. after this value is reached, the wait time will not increase further by the multiplier.
debezium.sink.pubsub.retry.initial.rpc.timeout.ms	10000	Controls the timeout for the initial Remote Procedure Call
debezium.sink.pubsub.retry.rpc.timeout.multiplier	2.0	The previous RPC timeout is multiplied by this multipler to come up with the next RPC timeout value, until the max is reached
debezium.sink.pubsub.retry.max.rpc.timeout.ms	10000	The max timeout for individual publish requests to Cloud Pub/Sub.
debezium.sink.pubsub.wait.message.delivery.timeout.ms	30000	The max wait time for retrieve of publish requests results to Cloud Pub/Sub.

Property	Default	Description
debezium.sink.pubsub.address		The address of the pubsub emulator. Only to be used in a dev or test environment with the pubsub emulator. Unless this value is set, debezium-server will connect to a cloud pubsub instance running in a gcp project, which is the desired behavior in a production environment.

Injection points

The Pub/Sub sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	D
io.debezium.server.pubsub.PubSubChangeConsumer.PublisherBuilder	@CustomConsumerBuilder	A pr cu fig st P us se sa de to

Interface	CDI classifier	D
io.debezium.server.StreamNameMapper		C pl tic th de (t in ca to By th na us

Pub/Sub Lite

Google Cloud Pub/Sub Lite is a cost-effective alternative to Google Cloud Pub/Sub. Pub/Sub exposes a set of REST APIs and provides a (not-only) Java SDK that is used to implement the sink.

Property	Default	Description
debezium.sink.type		Must be set to `pubsublite`
debezium.sink.pubsublite.project.id	system-wide default project id	A project name or project id in which the target topics are created.
debezium.sink.pubsublite.region		Region where the topics are being created. Example `us-east1-b`.
debezium.sink.pubsublite.ordering.enabled	`true`	Pub/Sub Lite can optionally use a message key to guarantee the delivery of the messages in with the same key to the same partition. This feature can be disabled.
debezium.sink.pubsublite.null.key	`default`	Tables without primary key sends messages with `null` key. This is not supported by Pub/Sub Lite so a surrogate key must be used.

Property	Default	Description
debezium.sink.pubsublite.wait.message.delivery.timeout.ms	30000	The max wait time for retrieve of publish requests results to Cloud Pub/Sub.

Injection points

The Pub/Sub Lite sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier
io.debezium.server.pubsub.PubSubLiteChangeConsumer.PublisherBuilder	@CustomConsumerBuilder
io.debezium.server.StreamNameMapper	

HTTP Client

The HTTP Client will stream changes to any HTTP Server for additional processing with the original design goal to have Debezium act as a Knative Event Source. The HTTP Client sink supports optional JSON Web Token (JWT) authentication.

Property	Default	Description
debezium.sink.type		Must be set to `http`
debezium.sink.http.url		The HTTP Server U stream events to. Th also be set by defini `K_SINK` environme able, which is used b Knative source fram
debezium.sink.http.timeout.ms	60000	The number of seco wait for a response f server before timing (default of 60s)
debezium.sink.http.retries	5	The number retries b exception is thrown 5 times).
debezium.sink.http.retry.interval.ms	1000	The number of millis to wait before anoth tempt to send recor made after failure (d 1s).
debezium.sink.http.headers.prefix	X-DEBEZIUM-	Headers will be pref this value (defaults t DEBEZIUM-).
debezium.sink.http.headers.encode.base64	true	Header values will b base64 encoded (de true).
debezium.sink.http.authentication.type		Specifies the authen type to use. If missir authentication is use Currently, only JSON Token (JWT) authen (indicated by the val `jwt`) is supported.
debezium.sink.http.authentication.jwt.username		Specifies the userna JWT authentication.
debezium.sink.http.authentication.jwt.password		Specifies the passw JWT authentication.

Property	Default	Description
debezium.sink.http.authentication.jwt.url		Specifies the base U (e.g., http://myserver for JWT authenticati paths auth/authentica auth/refreshTok appended for the JW and authentication F requests.
debezium.sink.http.authentication.jwt.token_expiration		Requested duration utes) before the auth tion token expires.
debezium.sink.http.authentication.jwt.refresh_token_expiration		Requested duration utes) before the refr ken expires.

Apache Pulsar

Apache Pulsar is high-performance, low-latency server for server-to-server messaging. Pulsar exposes a REST APIs and a native endpoint provides a (not-only) Java client that is used to implement the sink.

Property	Default	Description
debezium.sink.type		Must be set to `pulsar`.
debezium.sink.pulsar.timeout	`0`	Configures timeout in milliseconds for sending a batch of messages to Pulsar and waiting for the producer to flush and persist all of them. By default it is set to `0` which means no timeout. Make sure that `maxPendingMessages` and `blockIfQueueFull` are configured properly on the producer.
debezium.sink.pulsar.client.*		The Pulsar module supports pass-through configuration. The client configuration properties are passed to the client with the prefix removed. At least `serviceUrl` must be provided.
debezium.sink.pulsar.producer.*		The Pulsar module supports pass-through configuration. The message producer configuration properties are passed to the producer with the prefix removed. The `topic` is set by Debezium.
debezium.sink.pulsar.null.key	`default`	Tables without primary key sends messages with `null` key. This is not supported by Pulsar so a surrogate key must be used.

Property	Default	Description
debezium.sink.pulsar.tenant	public	The target tenant used to deliver the message.
debezium.sink.pulsar.namespace	default	The target namespace used to deliver the message.

Injection points

The Pulsar sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
io.debezium.server.StreamNameMapper		Custom implementation maps the planned destination (topic) name into a physical Pulsar topic name. By default the same name is used.

Azure Event Hubs

Azure Event Hubs is a big data streaming platform and event ingestion service that can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching/storage adapters.

Property	Default	Description
debezium.sink.type		Must be set to `eventhubs`.
debezium.sink.eventhubs.connectionstring		Connection string required to communicate with Even The format is: `Endpoint=sb://<NAMESPACE>/;SharedAccessKe <ACCESS_KEY_NAME>;SharedAccessKey= <ACCESS_KEY_VALUE>`
debezium.sink.eventhubs.hubname		Name of the Event Hub
debezium.sink.eventhubs.partitionid		(Optional) The identifier of the Event Hub partition tha events will be sent to. Use this if you want all the char events received by Debezium to be sent to a specific tion in Event Hubs. Do not use if you have specified `debezium.sink.eventhubs.partitionkey`

Property	Default	Description
debezium.sink.eventhubs.partitionkey		(Optional) The partition key will be used to hash the e Use this if you want all the change events received by Debezium to be sent to a specific partition in Event H not use if you have specified debezium.sink.eventhubs.partitionid
debezium.sink.eventhubs.maxbatchsize		Sets the maximum size for the batch of events, in byt

Using partitions in EventHubs

By default, when neither of the optional `debezium.sink.eventhubs.partitionid` or `debezium.sink.eventhubs.partitionkey` properties are defined, the EventHubs sink will send events round-robin to all available partitions.

You can enforce all messages to be sent to a single, fixed, partition by setting the `debezium.sink.eventhubs.partitionid` property. Alternatively, you can use the `debezium.sink.eventhubs.partitionkey` property to specify a fixed partition key that EventHubs will use to route all events to a specific partition.

If you have more specific routing requirements you can use the Partition Routing transformer. Ensure that the number of partitions specified in the transformer's `partition.topic.num` setting is equal or less to the number of partitions available in your EventHubs namespace, so that events cannot be routed to non-existing partition IDs. As an example, to route all events to 5 partitions based on their source schema name, you can set the following in your application.properties:

```
# Uses a hash of `source.db` to calculate which partition to send the event to. Ensures all events from the same source schema are sent to the same partition.
debezium.transforms=PartitionRouter
debezium.transforms.PartitionRouter.type=io.debezium.transforms.partitions.PartitionRouting
debezium.transforms.PartitionRouter.partition.payload.fields=source.db
debezium.transforms.PartitionRouter.partition.topic.num=5
```

Injection points

The default sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
com.azure.messaging.eventhubs.EventHubProducerClient	@CustomConsumerBuilder	Custom configu of a EventHubProd used to send me

Redis (Stream)

Redis is an open source (BSD licensed) in-memory data structure store, used as a database, cache and message broker. The Stream is a data type which models a *log data structure* in a more abstract way. It implements powerful operations to overcome the limitations of a log file.

Property	Default	Description
debezium.sink.type		Must be set to redis.
debezium.sink.redis.address		An address, formatted as host:port, at wl target streams are provided.
debezium.sink.redis.db.index	0	A number in the range 0..15 used for selecti to work with. Default is database 0. This feat able for standalone Redis connections; Redi only database 0.
debezium.sink.redis.user		(Optional) A user name used to communicat
debezium.sink.redis.password		(Optional) A password (of respective user) u cate with Redis. A password must be set if a
debezium.sink.redis.ssl.enabled	false	(Optional) A Boolean value that specifies wh tions to Redis require SSL.
debezium.sink.redis.null.key	de-fault	Redis does not support the notion of data wi string will be used as key for records without
debezium.sink.redis.null.value	de-fault	Redis does not support the notion of null pay case with tombstone events. So this string w value for records without a payload.
debezium.sink.redis.batch.size	500	Number of change records to insert in a sing (Pipelined transaction).
debezium.sink.redis.retry.initial.delay.ms	300	Initial retry delay when encountering Redis c OOM issues. This value will be doubled upo won't exceed debezium.sink.redis.retry.max.dela

Property	Default	Description
debezium.sink.redis.retry.max.delay.ms	10000	Max delay when encountering Redis connec issues.
debezium.sink.redis.connection.timeout.ms	2000	Connection timeout for Redis client.
debezium.sink.redis.socket.timeout.ms	2000	Socket timeout for Redis client.
debezium.sink.redis.wait.enabled	false	Enables wait for replica. In case Redis is cor replica shard, this allows to verify that the da ten to the replica. For more information see I command.
debezium.sink.redis.wait.timeout.ms	1000	Timeout in milliseconds when waiting for rep positive value.
debezium.sink.redis.wait.retry.enabled	false	Enables retry on wait for replica failure.
debezium.sink.redis.wait.retry.delay.ms	1000	Delay of retry on wait for replica failure.
debezium.sink.redis.message.format	com-pact	The format of the message sent to the Redis Possible values are extended (newer forma compact (the until now, old format). Read m message format below.
debezium.sink.redis.memory.threshold.percentage	85	The sink will stop consuming records if the u percentage (out of Redis configured maxmem or equal to this threshold. If the configured va this threshold is disabled.
debezium.sink.redis.memory.limit.mb	0	If Redis maxmemory is not available or 0, t debezium.sink.redis.memory.thresho will apply to this value (if this value is positive 0 (disabled).

Message Format

We have seen above the `debezium.sink.redis.message.format` property which configures the message format in two ways which look like this in Redis:

- the `extended` format, using two pairs {1), 2)}={"key", "message key"} and {3), 4)}={"value", "message value"}:

```
1) 1) "1639304527499-0"
   2) 1) "key"
      2) "{\"schema\": {\"type\": \"struct\", \"fields\": [{\"type\": \"int32\",
\"optional\": false, \"field\": \"empno\"}], \"optional\": false, \"name\":
\"redislabs.dbo.emp.Key\"}, \"payload\": {\"empno\": 11}}"
      3) "value"
```

```
      4) "{\"schema\": {\"type\": \"struct\", \"fields\": [{\"type\":
\"struct\", \"fields\": [{\"type\": \"int32\", \"optional\": false, \"field\":
\"empno\"}, {\"type\": \"string\", \"optional\": true, \"field\": \"fname\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"lname\"}, {\"type\":
\"string\", \"optional\": true, \"field\": \"job\"}, {\"type\": \"int32\",
\"optional\": true, \"field\": \"mgr\"}, {\"type\": \"int64\", \"optional\":
true, \"name\": \"io.debezium.time.Timestamp\", \"version\": 1, \"field\":
\"hiredate\"}, {\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"sal\"},
{\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"comm\"},
{\"type\": \"int32\", \"optional\": true, \"field\": \"dept\"}], \"optional\":
true, \"name\": \"redislabs.dbo.emp.Value\", \"field\": \"before\"}, {\"type\":
\"struct\", \"fields\": [{\"type\": \"int32\", \"optional\": false, \"field\":
\"empno\"}, {\"type\": \"string\", \"optional\": true, \"field\": \"fname\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"lname\"}, {\"type\":
\"string\", \"optional\": true, \"field\": \"job\"}, {\"type\": \"int32\",
\"optional\": true, \"field\": \"mgr\"}, {\"type\": \"int64\", \"optional\":
true, \"name\": \"io.debezium.time.Timestamp\", \"version\": 1, \"field\":
\"hiredate\"}, {\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"sal\"},
{\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"comm\"},
{\"type\": \"int32\", \"optional\": true, \"field\": \"dept\"}], \"optional\":
true, \"name\": \"redislabs.dbo.emp.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\": false, \"field\":
\"schema\"}, {\"type\": \"string\", \"optional\": false, \"field\": \"table\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"change_lsn\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"commit_lsn\"},
{\"type\": \"int64\", \"optional\": true, \"field\": \"event_serial_no\"}],
\"optional\": false, \"name\": \"io.debezium.connector.sqlserver.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\": \"redislabs.dbo.emp.Envelope\"}, \"payload\":
{\"before\": {\"empno\": 11, \"fname\": \"Yossi\", \"lname\": \"Mague\",
```

```
\"job\": \"PFE\", \"mgr\": 1, \"hiredate\": 1562630400000, \"sal\":
\"dzWUAA==\", \"comm\": \"AYag\", \"dept\": 3}, \"after\": null, \"source\":
{\"version\": \"1.6.0.Final\", \"connector\": \"sqlserver\", \"name\":
\"redislabs\", \"ts_ms\": 1637859764960, \"snapshot\": \"false\", \"db\":
\"RedisConnect\", \"sequence\": null, \"schema\": \"dbo\", \"table\": \"emp\",
\"change_lsn\": \"0000003a:00002f50:0002\", \"commit_lsn\":
\"0000003a:00002f50:0005\", \"event_serial_no\": 1}, \"op\": \"d\", \"ts_ms\":
1637859769370, \"transaction\": null}}"
```

- and the `compact` format, using only one pair {1), 2)}={"message key", "message value"}:

```
1) 1) "1639304527499-0"
   2) 1) "{\"schema\": {\"type\": \"struct\", \"fields\": [{\"type\": \"int32\",
\"optional\": false, \"field\": \"empno\"}], \"optional\": false, \"name\":
\"redislabs.dbo.emp.Key\"}, \"payload\": {\"empno\": 11}}"
      2) "{\"schema\": {\"type\": \"struct\", \"fields\": [{\"type\":
\"struct\", \"fields\": [{\"type\": \"int32\", \"optional\": false, \"field\":
\"empno\"}, {\"type\": \"string\", \"optional\": true, \"field\": \"fname\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"lname\"}, {\"type\":
\"string\", \"optional\": true, \"field\": \"job\"}, {\"type\": \"int32\",
\"optional\": true, \"field\": \"mgr\"}, {\"type\": \"int64\", \"optional\":
true, \"name\": \"io.debezium.time.Timestamp\", \"version\": 1, \"field\":
\"hiredate\"}, {\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"sal\"},
{\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"comm\"},
{\"type\": \"int32\", \"optional\": true, \"field\": \"dept\"}], \"optional\":
true, \"name\": \"redislabs.dbo.emp.Value\", \"field\": \"before\"}, {\"type\":
\"struct\", \"fields\": [{\"type\": \"int32\", \"optional\": false, \"field\":
\"empno\"}, {\"type\": \"string\", \"optional\": true, \"field\": \"fname\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"lname\"}, {\"type\":
\"string\", \"optional\": true, \"field\": \"job\"}, {\"type\": \"int32\",
\"optional\": true, \"field\": \"mgr\"}, {\"type\": \"int64\", \"optional\":
true, \"name\": \"io.debezium.time.Timestamp\", \"version\": 1, \"field\":
\"hiredate\"}, {\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"sal\"},
{\"type\": \"bytes\", \"optional\": true, \"name\":
\"org.apache.kafka.connect.data.Decimal\", \"version\": 1, \"parameters\":
{\"scale\": \"4\", \"connect.decimal.precision\": \"19\"}, \"field\": \"comm\"},
{\"type\": \"int32\", \"optional\": true, \"field\": \"dept\"}], \"optional\":
true, \"name\": \"redislabs.dbo.emp.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\": false, \"field\":
\"schema\"}, {\"type\": \"string\", \"optional\": false, \"field\": \"table\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"change_lsn\"},
{\"type\": \"string\", \"optional\": true, \"field\": \"commit_lsn\"},
{\"type\": \"int64\", \"optional\": true, \"field\": \"event_serial_no\"}],
\"optional\": false, \"name\": \"io.debezium.connector.sqlserver.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\": \"redislabs.dbo.emp.Envelope\"}, \"payload\":
{\"before\": {\"empno\": 11, \"fname\": \"Yossi\", \"lname\": \"Mague\",
\"job\": \"PFE\", \"mgr\": 1, \"hiredate\": 1562630400000, \"sal\":
\"dzWUAA==\", \"comm\": \"AYag\", \"dept\": 3}, \"after\": null, \"source\":
{\"version\": \"1.6.0.Final\", \"connector\": \"sqlserver\", \"name\":
\"redislabs\", \"ts_ms\": 1637859764960, \"snapshot\": \"false\", \"db\":
\"RedisConnect\", \"sequence\": null, \"schema\": \"dbo\", \"table\": \"emp\",
\"change_lsn\": \"0000003a:00002f50:0002\", \"commit_lsn\":
\"0000003a:00002f50:0005\", \"event_serial_no\": 1}, \"op\": \"d\", \"ts_ms\":
1637859769370, \"transaction\": null}}"
```


You can read more about Redis Streams here.

Injection points

The Redis sink behavior can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classi-fier	Description
io.debezium.server.StreamNameMapper		Custom implementation maps the planned destination (topic) name into a physical Redis stream name. By de-fault the same name is used.

NATS Streaming

NATS Streaming is a data streaming system powered by NATS, and written in the Go programming language.

Property	Default	Description
`debezium.sink.type`		Must be set to `nats-streaming`.
`debezium.sink.nats-streaming.url`		URL (or comma separated list of URLs) to a node or nodes in the cluster formatted as `nats://host:port`.
`debezium.sink.nats-streaming.cluster.id`		NATS Streaming Cluster ID.
`debezium.sink.nats-streaming.client.id`		NATS Streaming Client ID.

Injection points

The NATS Streaming sink behavior can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
`io.nats.streaming.StreamingConnection`	`@CustomConsumerBuilder`	Custom configured instance of a `StreamingConnection` used to publish messages to target subjects.
`io.debezium.server.StreamNameMapper`		Custom implementation maps the planned destination (topic) name into a physical NATS Streaming subject name. By default the same name is used.

NATS JetStream

NATS has a built-in distributed persistence system called JetStream which enables new functionalities and higher qualities of service on top of the base 'Core NATS' functionalities and qualities of service.

Property	Default	Description
`debezium.sink.type`		Must be set to `nats-jetstream`.
`debezium.sink.nats-jetstream.url`		URL (or comma separated list of URLs) to a node or nodes in the cluster formatted as `nats://host:port`.
`debezium.sink.nats-jetstream.create-stream`		If true, a basic stream will be created.

Property	Default	Description
debezium.sink.nats-jetstream.subjects	*.*.*	A comma separated list of subjects, messaging channel names. Can contain wildcards like test.inventory.*
debezium.sink.nats-jetstream.storage	memory	Controls how the messages are saved in the stream. Can be memory or file.

If you need a more configurable stream, it can be created with nats cli. More about streams at: https://docs.nats.io/nats-concepts/jetstream/streams

Injection points

The NATS JetStream sink behavior can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
io.nats.client.JetStream	@CustomConsumerBuilder	Custom configured instance of a JetStream used to publish messages to target subjects.
io.debezium.server.StreamNameMapper		Custom implementation maps the planned destination (topic) name into a physical NATS JetStream subject name. By default the same name is used.

Apache Kafka

Apache Kafka is a popular open-source platform for distributed event streaming. Debezium Server supports publishing captured change events to a configured Kafka message broker.

Property	Default	Description
debezium.sink.type		Must be set to kafka.
debezium.sink.kafka.producer.*		The Kafka sink adapter supports pass-through configuration. This means that all Kafka producer configuration properties are passed to the producer with the prefix removed. At least bootstrap.servers, key.serializer and value.serializer properties must be provided. The topic is set by Debezium.

Injection points

The Kafka sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classi-fier	Description
`io.debezium.server.StreamNameMapper`		Custom implementation maps the original destination (topic) name into another Kafka topic. By default, the same name is used.

Pravega

Pravega is a cloud-native storage system for event streams and data streams. This sink offers two modes: non-transactional and transactional. The non-transactional mode individually writes each event in a Debezium batch to Pravega. The transactional mode writes the Debezium batch to a Pravega transaction that commits when the batch is completed.

The Pravega sink expects destination scope and streams to already be created.

Property	Default	Description
`debezium.sink.type`		Must be set to `pravega`.
`debezium.sink.pravega.controller.uri`	`tcp://localhost:9090`	The connection string to a Controller in the Pravega cluster.
`debezium.sink.pravega.scope`		The name of the scope in which to find the destination streams.
`debezium.sink.pravega.transaction`	`false`	Set to `true` to have the sink use Pravega transactions for each Debezium batch.

Injection points

Pravega sink behavior can be modified by custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classi-fier	Description
`io.debezium.server.StreamNameMapper`		Custom implementation maps the planned destination (stream) name into a physical Pravega stream name. By default the same name is used.

Infinispan

Infinispan is open-source in-memory data grid that offers rich set of caches types as well as cache stores. Due to very fast data access, Infinispan can be used, besides others, as a data source for various data processing and analytical tools.

The Infinispan sink expects that the destination cache is already defined and created within the Infinispan cluster.

Property	Default	Description
`debezium.sink.type`		Must be set to `infinispan`.
`debezium.sink.infinispan.server.host`		The host name of one of the servers of the Infinispan cluster (can be also a comma-separated list of servers).
`debezium.sink.infinispan.server.port`	11222	The port of the Infinispan server.
`debezium.sink.infinispan.cache`		The name of the (exiting) cache where the records will be stored.
`debezium.sink.infinispan.user`		(Optional) The user name used for connecting to Infinispan cluster.
`debezium.sink.infinispan.password`		(Optional) The password used for connecting to Infinispan cluster.

Injection points

The Infinispan sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
`org.infinispan.client.hotrod.RemoteCache`	`@CustomConsumerBuilder`	Custom instance of Hot Rod cache which will be used for connecting and sending events to the Infinspan cluster.

Apache RocketMQ

Apache RocketMQ is a distributed messaging and streaming platform with low latency, high performance and reliability, trillion-level capacity and flexible scalability. Debezium server supports publishing captured change events to a configured RocketMQ.

Property	Default	Description
`debezium.sink.type`		Must be set to `rocketmq` .
`debezium.sink.rocketmq.producer.name.srv.addr`		Name server address of Apache RocketMQ .
`debezium.sink.rocketmq.producer.group`		Producer group of Apache RocketMQ.
`debezium.sink.rocketmq.producer.max.message.size`	4M, Suggest less than 4 MB.	(Optional) Maximum number of bytes of sent message body.
`debezium.sink.rocketmq.producer.send.msg.timeout`	3000ms	(Optional) The send message timeout duration is the waiting time for local synchronous invocation of clients. Set a proper value based on the actual application to avoid long thread blocking time.
`debezium.sink.rocketmq.producer.acl.enabled`	false	(Optional) The configuration is used to enable access authorization.
`debezium.sink.rocketmq.producer.access.key`		(Optional) The access key used for connecting to Apache RocketMQ cluster .
`debezium.sink.rocketmq.producer.secret.key`		(Optional) The access secret used for connecting to Apache RocketMQ cluster .

Injection points

The RocketMQ sink behaviour can be modified by a custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
`org.apache.rocketmq.client.producer.DefaultMQProducer`	`@CustomConsumerBuilder`	Custom configured instance of a RocketMQ used to publish messages to target topic.
`io.debezium.server.StreamNameMapper`		Custom implementation maps the planned destination (stream) name into a RocketMQ topic name. By default the same name is used.

RabbitMQ Stream

RabbitMQ is an open source message broker, supporting multiple messaging protocols and can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements. RabbitMQ supports messages queues and streams. Debezium Server supports publishing captured change events to a configured RabbitMQ Stream.

Property	Default	Description
`debezium.sink.type`		Must be set to `rabbitmq`.
`debezium.sink.rabbitmq.connection.host`	localhost	Host of RabbitMQ server.
`debezium.sink.rabbitmq.connection.port`	5672	Port of RabbitMQ server.

Property	Default	Description
`debezium.sink.rabbitmq.connection.*`		The RabbitMQ module supports pass-through configuration. The connection configuration properties are passed to the RabbitMQ client with the prefix removed.
`debezium.sink.rabbitmq.ackTimeout`	30000	Defines the maximum time in milliseconds to wait a confirm from the broker after publishing a message.
`debezium.sink.rabbitmq.exchange`	*topic name*	(Optional) Exchange name to use when publishing messages.
`debezium.sink.rabbitmq.routingKey`	*empty string*	(Optional) Static routing key to use when publishing messages.
`debezium.sink.rabbitmq.autoCreateRoutingKey`	false	(Optional) If `true` the non-existing routing key is automatically created.
`debezium.sink.rabbitmq.routingKeyDurable`	true	(Optional) If `true` the target queue content will survive a RabbitMQ server restart.
`debezium.sink.rabbitmq.routingKeyFromTopicName`	false	(Optional) If `true` the routing key is used from topic name instead of a static value.
`debezium.sink.rabbitmq.deliveryMode`	2	(Optional) The way how the message is delivered to and stored on a RabbitMQ server • 1 - Non-persistent • 2 - Persistent
`debezium.sink.rabbitmq.null.value`	`default`	RabbitMQ does not support the notion of null payloads, as is the case with tombstone events. So this string will be used as value for records without a payload.

Injection points

RabbitMQ sink behavior can be modified by custom logic providing alternative implementations for specific functionalities. When the alternative implementations are not available then the default ones are used.

Interface	CDI classifier	Description
`io.debezium.server.StreamNameMapper`		Custom implementation maps the planned destination (stream) name into a RabbitMQ exchange name and (if enabled) into the routing key name. By default the same name is used.

RabbitMQ Native Stream

Since RabbitMQ 3.9, Streams were introduced to RabbitMQ, utilizing a new blazingly-fast protocol that can be used alongside AMQP 0.9.1. Streams are great for large fan-outs, replay & time travel, and large logs, all with very high throughput (million messages per second).

Debezium Server is enhanced to support publishing captured change events to native RabbitMQ Streams leveraging RabbitMQ Stream Java Client.

Property	Default	Description
`debezium.sink.type`		Must be set to `rabbitmqstream`.
`debezium.sink.rabbitmqstream.connection.host`	localhost	Host of RabbitMQ server.
`debezium.sink.rabbitmqstream.connection.port`	5552	Port of RabbitMQ Stream Protocol.
`debezium.sink.rabbitmqstream.connection.*`		The RabbitMQ module supports pass-through configuration. The connection configuration properties are passed to the RabbitMQ client with the prefix removed.
`debezium.sink.rabbitmqstream.ackTimeout`	30000	Defines the maximum time in milliseconds to wait a confirm from the broker after publishing a message.
`debezium.sink.rabbitmqstream.null.value`	`default`	RabbitMQ does not support the notion of null payloads, as is the case with tombstone events. So this string will be used as value for records without a payload.

Extensions

Debezium Server uses the Quarkus framework and relies on dependency injection to enable developer to extend its behaviour. Note that only the JVM mode of Quarkus is supported, but not native execution via GraalVM. The server can be extended in two ways by providing a custom logic:

- implementation of a new sink
- customization of an existing sink - i.e. non-standard configuration

Implementation of a new sink

The new sink can be implemented as a CDI bean implementing interface `DebeziumEngine.ChangeConsumer` and with annotation `@Named` and unique name and scope `@Dependent`. The name of the bean is used as the `debezium.sink.type` option.

The sink needs to read the configuration using Microprofile Config API. The execution path must pass the messages into the target system and regularly commit the passed/processed messages.

See the Kinesis sink implementation for further details.

Customization of an existing sink

Some of the sinks exposes dependency injections points that enable users to provide its own bean that would modify the behaviour of the sink. Typical examples are fine tuning of the target client setup, the destination naming etc.

See an example of a custom topic naming policy implementation for further details.

Cassandra connector

Running Debezium Server with Cassandra connector

Running with java 11+ requires setting the following java options at startup trough the JDK_JAVA_OPTIONS environment variable or equivalent:

```
JDK_JAVA_OPTIONS="--add-exports java.base/jdk.internal.misc=ALL-UNNAMED --add-exports
java.base/jdk.internal.ref=ALL-UNNAMED --add-exports java.base/sun.nio.ch=ALL-UNNAMED --add-
exports java.management.rmi/com.sun.jmx.remote.internal.rmi=ALL-UNNAMED --add-exports
java.rmi/sun.rmi.registry=ALL-UNNAMED --add-exports java.rmi/sun.rmi.server=ALL-UNNAMED --
add-exports java.sql/java.sql=ALL-UNNAMED  --add-opens java.base/java.lang.module=ALL-
UNNAMED --add-opens java.base/jdk.internal.loader=ALL-UNNAMED --add-opens
java.base/jdk.internal.ref=ALL-UNNAMED --add-opens java.base/jdk.internal.reflect=ALL-
UNNAMED --add-opens java.base/jdk.internal.math=ALL-UNNAMED --add-opens
java.base/jdk.internal.module=ALL-UNNAMED --add-opens java.base/jdk.internal.util.jar=ALL-
UNNAMED --add-opens jdk.management/com.sun.management.internal=ALL-UNNAMED --add-
opens=java.base/java.io=ALL-UNNAMED --add-opens java.base/sun.nio.ch=ALL-UNNAMED"
```

Sample of basic `application.properties` for running Cassandra connector with Redis sink

```
# Sink
debezium.sink.type=redis
debezium.sink.redis.address=localhost:6379
```

```
# Connector
debezium.source.connector.class=io.debezium.connector.cassandra.Cassandra4Connector
## node.id must be unique per each connector running on each Cassandra node
debezium.source.cassandra.node.id=sample_node_01
debezium.source.cassandra.hosts=127.0.0.1
debezium.source.cassandra.port=9042
debezium.source.cassandra.config=/opt/cassandra/conf/cassandra.yaml
debezium.source.commit.log.relocation.dir=cassandra/relocdir
debezium.source.offset.storage=io.debezium.server.redis.RedisOffsetBackingStore
debezium.source.topic.prefix=sample_prefix
## internal Cassandra http port
debezium.source.http.port=8040
```

Transformation for Operation Code

By default, Cassandra connector has it's own Operation Codes which are not entirely compatible with Debezium Operation Codes. If needed, a specific transform can be defined in Debezium Server's `application.properties` to enable the conversion:

```
debezium.transforms=EnvelopeTransformation
debezium.transforms.EnvelopeTransformation.type=io.debezium.connector.cassandra.transforms.Enve
```

This will convert Operation Codes as follows:

```
INSERT "i"          -> CREATE "c"
UPDATE "u"          -> UPDATE "u"
DELETE "d"          -> DELETE "d"
RANGE_TOMBSTONE "r" -> TRUNCATE "t"
```