Redis Connect

Version 0.10.3

Table of Contents

Introduction . . . 1
Key Terms . . . 2
Source Distributions . . . 3
Supported Sources . . . 4
Redis Connect Management . . . 5
Production Deployment . . . 8
Configuration Overview . . . 17
Cluster Configuration . . . 20
Job Configuration . . . 25
Database Configuration . . . 36
Custom Database Properties . . . 38

Introduction

Redis Connect is a distributed platform that enables real-time event streaming, transformation, and propagation of changed-data events from heterogeneous data platforms to Redis Stack, Redis Cloud, and Redis Enterprise.

Redis Connect effectively captures change data events from source databases and writes that data to Redis. This allows you to keep a Redis database in sync with a variety of source databases. You can then use Redis to serve this data to downstream applications at low latencies.

This document outlines key terms, installation instructions, production readiness guidelines, and definitions for the various configuration directives for Redis Connect clusters and jobs[1].

Table of Contents

- Key Terms
- Source Distributions
- Production Deployment
- Configuration Overview
- Cluster Configuration
- Job Configuration
- Database Configuration
- Custom Database Properties

[1] Some conventions used in this document include the following. **Property names**: `property.name`. **Name formats**: `Name Format`. **Warnings**: `Warning`

Key Terms

To understand how Redis Connect works, it's important to review some key terms:

Source

A database, such as PostgreSQL, whose data will be replicated to Redis. Redis Connect replicates data from a source database to Redis.

Target

A database to write data to. With Redis Connect, the target is usually Redis.

Job

A stream of change-data events replicating from source to target. For example, you can replicate all changes from a given set of PostgreSQL tables to Redis and maintain consistency between these tables and their Redis representations in real time.

Job Types

Redis Connect supports two types of jobs: **initial load jobs** and **stream jobs**.

- **Initial load jobs** create a point-in-time snapshot of the tables to be replicated and then transfer their data to Redis.
- **Stream jobs** (also known as CDC or "change data capture" jobs) replicate changes from the source tables to Redis as those changes occur.

Instance

A single JVM process running Redis Connect. Because Redis Connect is a distributed platform, it may run as one or more coordinated instances.

Cluster

One or more **instances** of Redis Connect running in a coordinated fashion.

Partition

A way of dividing jobs to scale them horizontally. Each job may be divided into one or more partitions. Partitions are divided automatically among Redis Connect instances. If a Redis Connect instances becomes unavailable, all jobs partitions will be migrated to another available instance.

Source Distributions

Redis Connect releases are distributed on Github.

See the Redis Connect Release History to download the latest distributions of Redis Connect.

The source distribution contains four relevant folders:

lib

JARs for Redis Connect and its dependencies.

extlib

JARs for Redis Connect custom stages and database drivers not including the Redis Connect distribution (e.g., Oracle and DB2). These database drivers must be provided by the Redis Connect user.

config

Working configuration files, sample payloads for configuring jobs, and Grafana dashboard configurations.

bin

Scripts for running Redis Connect on Linux VMs, in container environments, and on Windows.

Supported Sources

Redis Connect can capture change data from several RDBMSMS and NoSQL databases. Redis Connect can also load data from CSV and JSON files.

The table below show the complete list of supported sources. Each source database name links to a demo with sample configuration and data.

Under the hood, Redis Connect uses Debezium to access most of its source databases. In production, it's important that your source database is configured as Debezium expects. Be sure to review the Debezium doc links below.

Table 1. Supported source databases

Database name	Job types	Debezium Docs
DB2	Initial load	Debezium docs
Files	Initial load	n/a
Gemfire	Initial load & Stream	n/a
MongoDB	Initial load & Stream	Debezium docs
MySQL	Initial load & Stream	Debezium docs
Oracle	Initial load & Stream	Debezium docs
PostgreSQL	Initial load & Stream	Debezium docs
Splunk	Initial load	n/a
SQLServer	Initial load & Stream	Debezium docs
Vertica	Initial load	n/a

Redis Connect Management

Once Redis Connect is installed and running, you manage Redis Connect using its REST API or command line interface (CLI).

REST API

Each Redis Connect instance can be configured to expose a REST API with a Swagger interface for ease of use.

The table below shows a few of the commonly-used REST endpoints. For the complete REST API documentation, see the Redis Connect Swagger API Docs.

Table 2. Common REST API Endpoints

Endpoint name	Description	Documentation
Create job	Saves job configuration for the provided `jobName`. Many of the optional job configuration attributes have default values which can be reviewed once the job configuration is saved or found in the documentation. Example Redis key: {connect}:job:config:jobName	Create job Swagger endpoint

Endpoint name	Description	Documentation
Start job	Starts a job, including all job partitions. This includes both initial load and stream jobs. For a job start, all job partitions must be stopped or never before started. There is no guarantee on which cluster instance will claim a job partition, and there is no advantage to initiating this operation from a specific cluster member. Before a start is initiated, a job configuration must be created and a validated to confirm enough remaining capacity exists across the cluster for all job partitions (this does not apply to initial load).	Start job Swagger endpoint
Stop job	Stops a job, including all job partitions. You cannot stop initial load jobs since they are removed automatically upon completion. For a job to be stopped, all job partitions must be active. Job claims, metrics, and checkpoints will all be preserved upon a job stop. This ensures that the job can later resume from where it was stopped.	Stop job Swagger endpoint
Job claim status	Returns job claims from across the cluster that match the requested jobStatus. Valid job statuses are: `staged`, `stopped`, and `all`. Use this endpoint to see which jobs the cluster is managing, and their status.	Job claim status Swagger endpoint

REST API Security

In a production environment, the Swagger API may require an open port in a firewall. By default, the API is available on port 8282, but this is configurable.

If you are running multiple Redis Connect instances on the same server, each instance will require a different port for its REST API.

CLI

Redis Connect includes a command line interface that exposes the management functions provided by the REST API. You can start a CLI instance from the command line as follows:

```
$ ./bin/redisconnect.sh cli
```

Once the CLI has started, type `help` to see the available commands.

Production Deployment

Redis Connect is deployed as one or more JVM instances coordinated as a cluster. Below are recommendations for running Redis Connect in production.

Environment

Redis Connect can be deployed on physical servers, virtual machines, or using Docker or any Kubernetes-based environment.

The minimum resource requirements per Redis Connect instance are as follows:

- 4 CPU cores
- 2 GB memory
- 20 GB of free disk space
- 1 Gbps networking

We recommend allocating one thread per job partition. You can deploy more than one Redis Connect instance on a single VM. The number of Redis Connect instances that can effectively be deployed on a single machine or VM will depend on that VM's physical memory and number of CPU cores. However, for high availability, you must deploy your Redis Connect instances across more than one physical server or VM. See High Availability below for more details.

Operating System and JVM

Redis Connect can run on any operating system hosting a Java runtime environment. However, for production deployments, we recommend Linux.

Redis Connect is supported on Java versions 11 and greater.

For information on deploying to Kubernetes environments, see the Redis Connect Kubernetes documentation.

Environment Variables

Redis Connect recognizes and depends upon several environment variables. You can see example of these in the startup scripts included in the Redis Connect distribution.

- `REDISCONNECT_MIN_JAVA_VERSION="11"`
- `REDISCONNECT_HOME="${REDIS_CONNECT_HOME_DIR}"`
- `REDISCONNECT_JOB_MANAGER_CONFIG_PATH="$REDISCONNECT_HOME/config/jobmanager.properties"`
- `REDISCONNECT_LOGBACK_CONFIG="$REDISCONNECT_HOME/config/logback.xml"`
- `REDISCONNECT_LOGBACK_CLI_CONFIG="$REDISCONNECT_HOME/config/logback-cli.xml"`
- `REDISCONNECT_JAVA_OPTIONS="-XX:+HeapDumpOnOutOfMemoryError -Xms1g -Xmx2g"`
- `REDISCONNECT_EXTLIB_DIR="$REDISCONNECT_HOME/extlib"`

- `REDISCONNECT_LIB_DIR="$REDISCONNECT_HOME/lib/*:$REDISCONNECT_EXTLIB_DIR/\*"`

JVM Flags

We recommend the following JVM flags for each Redis Connect JVM instance:

- Enable heap dump on OOM: `-XX:+HeapDumpOnOutOfMemoryError`
- Min heap size of 1 GB: `-Xms1g`
- Max heap size of 2 GB: `-Xmx2g`

To set Redis Connect's JVM options, use the `REDISCONNECT_JAVA_OPTIONS` environment variable:

```
REDISCONNECT_JAVA_OPTIONS="-XX:+HeapDumpOnOutOfMemoryError -Xms1g -Xmx2g"
```

Redis Requirements

Redis Connect requires a working Redis Enterprise Software or Redis Cloud installation.

We recommend provisioning two Redis databases. These databases can reside in the same Redis Enterprise cluster.

The first database will serve as your Redis Connect cluster's configuration store. This database should be configured as follows:

- Data persistence enabled (RDF + AOF every second)
- High availability enabled
- ACLs enabled (See Security below)

The second database will serve as the target for replication from the source database. Since this is an operational database receiving change-data events, the sizing for this database depends on the sizes of the tables being replicated and on the volume of change-data events. In all cases, we still recommend:

- High availability enabled
- ACLs enabled (See Security below)

Logging

Redis Connect uses Logback for logging. See your Redis Connect distribution's `config/logback.xml` for a sample Logback configuration file.

Redis connect locates its logging config files using the following environment variables:

- `REDISCONNECT_LOGBACK_CONFIG="$REDISCONNECT_HOME/config/logback.xml"`
- `REDISCONNECT_LOGBACK_CLI_CONFIG="$REDISCONNECT_HOME/config/logback-cli.xml"`

Redis Connect has been designed to provide descriptive logs to make troubleshooting easier.

If you need to change your application log level at runtime, you can do this using the REST API. See the loglevel REST endpoint documentation for details.

Note that log level changes are not global; they apply only to the instance whose REST API you are connected to. To change the log level for a given instance, connect directly to that instance's REST API.

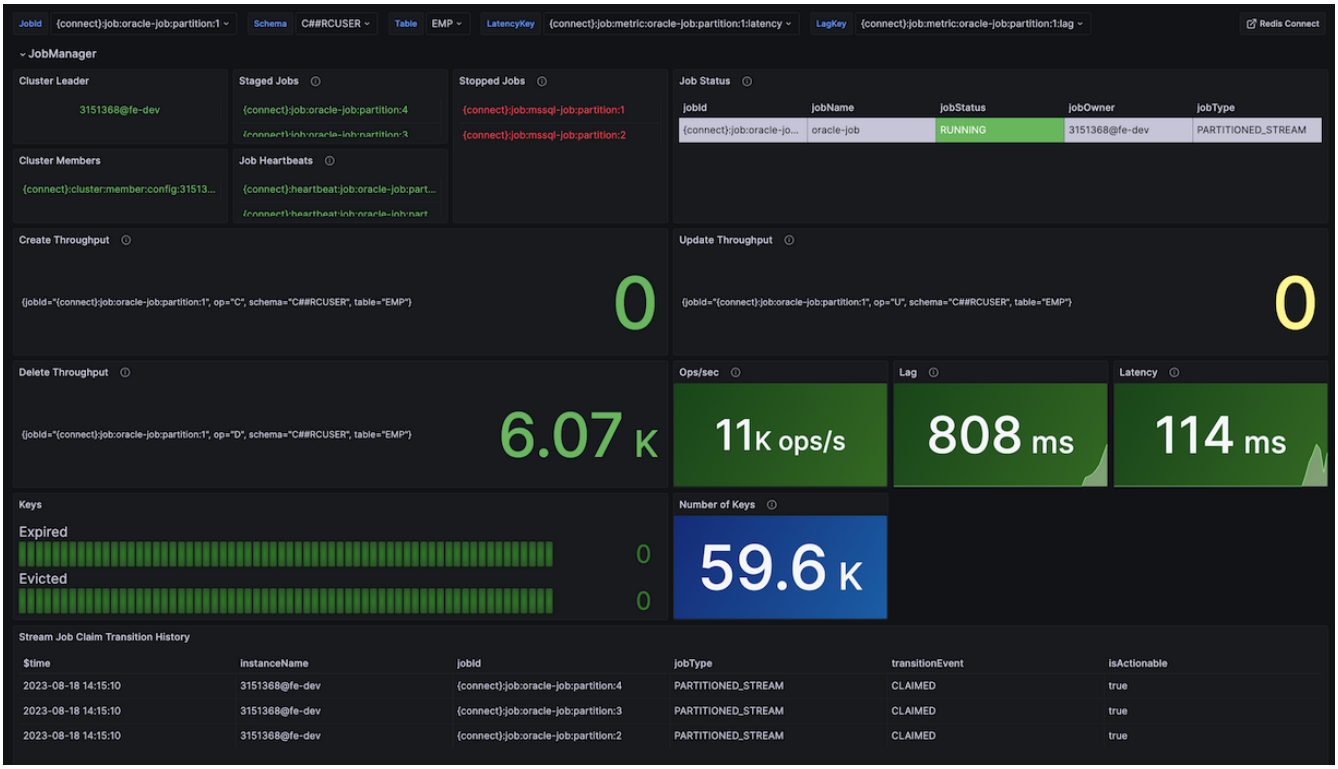
Monitoring

Redis Connect publishes performance metrics to job manager's configuration database. This takes advantage of Redis' time series capabilities.

You can view these metrics in Grafana using the Redis Datasource for Grafana.

Your Redis Connect distribution includes a pre-configured dashboard for viewing key operational metrics. You can find this dashboard and its Grafana configuration at `config/samples/dashboard`, and this configuration can be modified to fit your monitoring requirements.

The sample dashboard looks like this:



Important metrics include the following:

Staged jobs

The list of all configured jobs in the Redis Connect cluster.

Job heartbeats

Heartbeats for the currently active job partitions. Each active partition records a heartbeat every second, indicating that the job is actively being run by a Redis Connect instance.

Ops per second

The number of write operations against the target database completed per second.

Lag

The average elapsed time between the moment a change event is published to the source database and the moment that event is written to the target database (i.e., Redis). High lag values may indicate that your Redis Connect cluster is failing to keep up with the volume of CDC changes. Note that this metric only applies to stream jobs.

Latency

The average amount of time, in milliseconds, that it takes to publish a change event from the source database to Redis.

Security

Redis Connect should be run in a secure network environment. Because Redis Connect connects to source and target databases, those connections **may** be secured with TLS/SSL and **must** be secured using authentication credentials.

To support additional security requirements, Redis Connect supports file-based credential rotation, optionally powered by secrets management frameworks like Hashicorp Vault.

TLS/SSL Support

Most database systems support one-way and mutual TLS, and TLS authentication is often required in production environments.

Redis Connect manages certificate-based authentication using Java's KeyStore and TrustStore support. To configure the Java KeyStore and TrustStore, see the KeyStore and TrustStore configuration reference.

To configure Redis Enterprise with TLS, see the documentation:

- TLS with Redis Cloud
- TLS with Redis Enterprise Software

Database ACLs

The Redis job manager and the job target databases should have ACLs enabled. To keep your Redis databases secure and prevent accidental misconfigurations, we recommend the following policies:

1. Create separate Redis database users for your Redis Connect instances, Redis Connect administrators, and developers.
2. Developers should be provided with read-only access to these Redis databases.
3. Ensure that any application user connecting Redis Connect's Redis databases has `dangerous` commands disabled.
4. Users should not have permission to delete keys starting with "{connect}". This prevents

accidental deletion of important configuration.

Authentication Credentials

Redis Connect gets its database authentication credentials from properties files stored on the filesystem.

The file `redisconnect_credentials_jobmanager.properties` is common to all Redis Connect instances. This file provides the following authentication credentials:

- Username and password for the Redis database used to store Redis Connect's configuration
- Password for the certificate TrustStore (when applicable)
- Password for the certificate KeyStore (when applicable)
- Password for an SMTP mail server, when email alerts are enabled

In addition to the job manager's credentials, Redis Connect requires two credential files for each job: one for the source database and another for the target Redis database.

Source database credential files are named according to this scheme:

```
redisconnect_credentials_[SOURCE_DB_NAME]_[JOB_NAME].properties
```

For example, if you have a job where PostgreSQL is the source database, and the job is called "user-replication", then you will need a properties file called:

```
redisconnect_credentials_PostgreSQL_user-replication.properties
```

The **target database credentials files** are named as follows:

```
redisconnect_credentials_redis_[JOB_NAME].properties
```

For job called "user-replication", the Redis target database credentials file will be named:

```
redisconnect_credentials_redis_user-replication.properties
```

You can see examples of credentials files in the `config/samples/credentials` directory of the Redis Connect distribution.

Securing Credentials Files

Because credentials files store sensitive information in plain text on the filesystem, these files must have strict permissions. Only authorized users should be able to read and write to these files.

As with private key files, these credentials files should be owned by the redis connect user be set

read/write only (e.g., chmod 600).

Credential Rotation

Redis Connect instances listen for changes to the credentials files. When the contents of these files changes, Redis Connect will read the changes to ensure that database connectivity is maintained.

To enable support for credential rotation in Redis Connection, open `jobmanager.properties` and ensure that the credential rotation directives are uncommented. The following configuration enables the credential file listener and sets it to check for changes every 60 seconds:

- `credentials.rotation.event.listener.enabled=true`
- `credentials.rotation.event.listener.interval=60000`

Secrets Management

You can use a secrets management framework such as Hashicorp Vault to securely store and rotate credentials. To use a secrets management framework:

1. Ensure that framework can write property files in a given mounted path on your filesystem.
2. In `jobmanager.properties`, set the `credentials.dir.path` to this mounted path.

High availability

For high-availability, we recommend that you employ n+1 redundancy as part of your Redis Connect cluster. In this case, n is defined as the minimum number of Redis Connect instances required to provision all of the jobs that you intend to run.

We also recommend that you deploy each Redis Connect instance on a separate VM. When running in the cloud, you should ensure that your Redis Connect instances are distributed evenly across cloud availability zones.

Instance failure behavior

In the event of a Redis Connect instance (JVM) failure, the heartbeat lease of each job partition owned by that instance expires. Once expired, the job reaper will identify each partition without a heartbeat and immediately publish a job claim request.

Even if a JVM was restarted on the same machine, on which it just failed, it would still need to compete with the other instances with available capacity to claim the job partitions.

It's important to understand each job's capacity requirements and related settings to calculate how many instances are needed. For example, if a job with 4 partitions and a setting of `maxPartitionsPerClusterMember=2` is deployed across a 2-instance cluster, then even if there are 20 available capacity on each Redis Connect instance, they would be blocked from claiming more of this job's unclaimed partitions in the event of a node failure.

It's also important to set the Job Manager property `job.claim.max.capacity` appropriate to the desired redundancy requirement with respect to all job partitions planned for deployment across

the cluster. For example, if there are 3 cluster nodes each with a goal to deploy a job with 4 partitions, then in order to achieve n+1 redundancy, each node would have to configure job.claim.max.capacity=2, or more, since the loss of one node would still allow for all 4 partitions to be claimed.

Instance failures

For high-availability, it is recommended to deploy each Redis Connect instance on a separate VM or cloud availability zone. For on-premises deployments, extra care should be taken to ensure that each VM is deployed on different underlying server racks to avoid a single point of failure.

Each VM / cloud availability zone should have access to each job's source, job's target, and the Redis Connect Job Manager database.

For deployment on Kubernetes, see the Redis Connect Kubernetes Docs.

Network partitions

If a network partition occurs, then one side or the other (or both) of the partition needs to stop responding to requests to maintain the consistency guarantee. If both sides continue to respond to reads and writes while they are unable to communicate with each other, they will diverge and no longer be consistent. This state where both sides of the partition remain available is called "split brain".

To avoid "split brain", Redis Connect jobs make a call to the job manager database as the initial stage of their producer's source event-loop to check if their claim is still valid. Here's why that matters. A job cannot be claimed unless the Job Reaper identifies a staged job without a heartbeat, so until that lease expires, the claim is intact. Once the heartbeat expires due to the network partition, the Job Reaper will first remove the claim from the job manager database before publishing a new job claim request. So even if the original Redis Connect instance has not died, it cannot continue to process additional batches once its claim is removed nor can it regain its ownership by default when the network partition is remedied.

By relying on Redis Enterprise as its source of truth, this cluster architecture is guaranteed to always have a single job owner at any one point-in-time. This is because the metadata stored within Redis Enterprise has its own split-brain protection.

Source Connection Failures

Connections fail for many reasons, both intermittent drop-outs and prolonged outages. Since Redis Connect does not know the cause of a connection issue, the cluster will always assume that the connection failure is intermittent and attempt to re-establish the connection.

For this purpose, it's important to set the following configurations within the job source:

- `sourceConnectionMaxRetryAttempts`
- `sourceConnectionMaxRetryDuration`
- `sourceConnectionRetryDelayInterval`

- `sourceConnectionRetryMaxDelayInterval`
- `sourceConnectionRetryDelayFactor`

Here's an example of their use:

- `sourceConnectionMaxRetryAttempts=3`
- `sourceConnectionMaxRetryDuration=10`
- `sourceConnectionRetryDelayInterval=60`
- `sourceConnectionRetryMaxDelayInterval=600`
- `sourceConnectionRetryDelayFactor=3`

Based on the above, if the first attempt to connect fails, then the process waits 60 seconds before attempting its first retry.

If the retry fails, then the process waits 60 * 3 = 180 seconds for its second retry, and so on, up until the max delay interval between retries.

Target Redis Connection Failures

Just like sources, Redis target connections have a retry mechanism. However, these retries are built into the underlying Redis client (a.k.a., Lettuce). For production environments, we recommend using the default Redis connection settings.

Target Redis Slow Consumer

It's possible that the rate of incoming changed-data events and outgoing sink commits becomes unbalanced due to a slowdown on the target database, usually because of network congestion. To avoid overwhelming the job pipeline queue, the combination of the following settings helps to provide back-pressure support:

- slowConsumerMaxRetryAttempts
- intermittentEventSleepDuration

Once the pipeline queue becomes full, a circuit breaker will be triggered to disable the producer's polling event-loop until the requisite pipeline capacity becomes available. While the pipeline is iterating through its queued events, the job's producer will periodically attempt to publish the remainder of its current batch.

In between each attempt, the producer will sleep (based on intermittentEventSleepDuration), which allows the pipeline time to catch up. This loop will continue until either the required capacity becomes available or the `slowConsumerMaxRetryAttempts` limit is reached. Once reached, the job will stop.

In production, we recommend that you set `slowConsumerMaxRetryAttempts` to a reasonable setting instead of making it unlimited or disabling it. This is where proper capacity estimation and partitioning will become critical since you don't want to stop the job unnecessarily due to a network disruption nor large spike in volume. In other words, even if a single partition can handle

the entire throughput on a regular basis, it might be worthwhile to partition the deployment so that a sudden spike in throughput or a slow consumer does not bring down the job.

Recovery Time Considerations

Redis Connect's job management processes are scheduled threads that periodically wake up to perform their service. While it's not critical to understand the internals of the cluster architecture, it is relevant to be aware of the balance between job claim heartbeats and the Job Reaper service.

As mentioned previously, when a job partition is claimed it begins a heartbeat lease which has a TTL set by `job.claim.heartbeat.lease.renewal.ttl`. If the lease cannot be renewed before expiry, the Job Reaper service will remove the current owner's claim based on the assumption that its Redis Connect instance is no longer alive.

This service is performed based on a fixed interval set by job.reap.attempt.interval. Once the job claim is removed, each eligible Redis Connect instance will attempt to claim the job partition. This Job Claim service is also based on a fixed interval set by job.claim.attempt.interval.

Since each interval is started at a different point-in-time (and some services have more work to do than others), it's impossible to synchronize all the moving parts. However, it is critical to avoid scenarios that can cause unexpected behavior by giving enough buffer between associated services. For example, `job.reap.attempt.interval` must be at least 2 seconds more than `job.claim.attempt.interval` to avoid sending multiple job claim requests for the same partition. This relates to recovery because each default interval can potentially add up to over a minute in recovery time inclusive of the actual start job process. Therefore, if instant recovery is critical to production SLAs, our recommendation is to consider the balance between separate services and as well as the stability of your network. If these settings are set too low, it's possible that a network disruption can unnecessarily start the recovery process on a healthy job.

See Cluster Configuration for details on setting these properties.

Configuration Overview

A Redis Connect cluster consists of one more JVM instances. These instances coordinate to process configured jobs.

Every Redis Connect instances has a main configuration file called `jobmanager.properties`. You can find an example of this file in your distribution's "/config" directory.

The location of this file is provided to each Redis Connect instance with the `REDISCONNECT_JOB_MANAGER_CONFIG_PATH` environment variable. Here's an example of its use:

- `REDISCONNECT_JOB_MANAGER_CONFIG_PATH="$REDISCONNECT_HOME/config/jobmanager.properties"`

Cluster Instance Configuration

The most basic Redis Connect instance configuration includes the URL of the Redis database used to store the cluster's state. This URL is specified with the property `redis.connection.url` in `jobmanager.properties`.

Here's a sample `jobmanager.properties` file:

```
#####     Cluster properties                                    ①
cluster.name=default

#####     Job Manager Database properties                       ②
redis.connection.url=redis://127.0.0.1:14001
redis.connection.auto.reconnect=true

#####     Credentials properties                                ③
credentials.dir.path=/usr/local/redis-connect/credentials
credentials.rotation.event.listener.enabled=true

#####     REST properties                                       ④
rest.api.enabled=true
rest.api.port=8282

#####     Job Manager Services properties                       ⑤
job.manager.services.enabled=true
job.manager.services.threadpool.size=2
```

① The **cluster properties** define general Redis Connect cluster configuration.

② The **job manager database properties** determine which database Redis Connect uses to store its configuration.

③ The **credentials properties** tell Redis Connect where to find credentials files.

④ The **REST properties** determine whether the REST API is enabled and how it runs.

⑤ The **job manager services properties** lets you specify the resources to allocate for job managements on this JVM.

Job Configuration

Redis Connect runs one or more **change-data capture jobs**. Each job reads data from a source database and ultimately writes that data to a Redis target database. A job is defined by a JSON document. To create a job, you submit this JSON document using the REST API or CLI. Once a job is submitted, the Redis Connect cluster leader will schedule the job among available instances.

The following section shows how to construct a JSON-based configuration for a job. To see several real-world Job configurations, open the `config/samples/payloads` folder in the Redis Connect distribution.

Job configuration format

Every job configuration has at least three top-level keys: partitions, source, and pipeline.

```
{
  "partitions" : NUMBER_OF_PARTITIONS,    ①
  "source": { SOURCE_JSON_DEFINITION },   ②
  "pipeline": { TARGET_JSON_DEFINITION }  ③
}
```

① The **partitions** field defines the level of parallelism to use when running the job.

② The **Job Source Properties** field stores configuration for the job's source database (e.g., MySQL, Oracle, etc.)

③ The **Job Pipeline Properties** field stores configuration for the target of the job. This includes custom transformations and the target Redis database.

Here's a more detailed job configuration with some of these valued filled in:

```
{
  "partitions" : 1,
  "source": {
    "database": {                     ①
      "databaseType": "POSTGRES",
      "databaseURL": "jdbc:postgresql://127.0.0.1:5432/RedisConnect",
      "customConfiguration": {        ⑤
        "database.dbname" : "RedisConnect",
      }
    },
    "tables": {                       ②
      "public.emp": {
        "columns": [
          { "targetColumn": "empno", "sourceColumn": "empno", "targetKey": true},
          { "targetColumn": "hiredate", "sourceColumn": "hiredate", "type":
"DATE_TIME"}
        ],
        "initialLoad": {
          "partitions": 4
```

```
        },
        "autoConfigColumnsEnabled": true
      }
    }
  },
  "pipeline": {
    "stages": [                          ③
      {
        "index": 1,
        "stageName": "REDIS_HASH_SINK",
        "database": {                    ④
          "credentialsDirectoryPath" : "../config/samples/credentials",
          "databaseURL": "redis://127.0.0.1:14000",
          "databaseType": "REDIS",
          "customConfiguration": {       ⑤
            "redis.connection.sslEnabled": false,
            "truststore.file.path": "../config/samples/credentials/client-
truststore.jks"
          }
        }
      }
    ]
  }
}
```

① A **database configuration** for the database that Redis Connect reads from.

② A **tables configuration** specifying which tables to read from.

③ The list of **stages**, which always ends with the Redis target database.

④ The **database configuration** for the target database. The final stage in the pipeline is always the Redis target database.

Note that all **database configuration** objects include a **customConfiguration** field. <5> The **customConfiguration** field lets you pass in database-specific configurations. For example, Postgres has a `database.dbName` field that is unique to Postgres.

Submitting a job configuration

To save a job configuration, construct your JSON payload and then submit it using the REST API's create job endpoint. When you submit this configuration, you will also provide a unique job name. This job name will be used in the naming of your credentials files.

Credentials files

Credentials files store authentication credentials for the databases used by Redis Connect.

The location of these credentials files is specified at the `credentials.dir.path` properties in `jobmanager.properties`. To learn about which credentials are required, see the authentication credentials subsection.

Cluster Configuration

This section describes the configuration directives in `jobmanager.properties`. Each of the subsections below corresponds to a configuration block in `jobmanager.properties`. To see a sample `jobmanager.properties` file, open `config/jobmanager.properties` in your Redis Connect distribution.

Cluster properties

Table 3. Cluster properties

Property name	Type	Description	Default
cluster.name	String	Metadata purposes only. Non-functional	default
cluster.leader.heartbeat.lease.renewal.ttl	Integer	TTL (Time-to-Live) which is renewed upon each `cluster.election.attempt.interval` iteration by the cluster leader. Measured in milliseconds with a minimum of 1 second (1000 ms).	5000
cluster.election.attempt.interval	Integer	Fixed rate scheduled thread which either renews or elects a new cluster leader. Runs on each Redis Connect Instance (JVM) when `job.manager.services.enabled=true`. Measured in milliseconds with a minimum of 1 second (1000 ms)	5000
cluster.timeseries.metrics.enabled	Boolean	Enables creation of a scheduled thread for job metrics reporting to RedisTimeSeries within the job manager database.	false

Job manager services properties

Table 4. Job manager services properties

Property name	Type	Description	Default
job.manager.services.enabled	Boolean	Enables creation of scheduled thread(s) to participate in cluster leader elections, facilitate REST API / CLI (Job Manager service), and identify staged jobs without a heartbeat lease (Job Reaper service). When this property is disabled, the Redis Connect instance may still participate in job execution and job claim attempts (Job Claimer service).	true

Property name	Type	Description	Default
job.manager.services.threadpool.size	Integer	For non-production deployments, one thread is adequate. In production, we recommend two threads.	2
job.reap.attempt.interval	Integer	The interval between attempts to identify staged jobs without a heartbeat lease. Implemented as a scheduled thread that runs on each Redis Connect Instance (JVM) when `job.manager.services.enabled=true`. Measured in milliseconds with a minimum of 1 second (1000 ms)	7000
job.claim.service.enabled		Enables creation of scheduled thread(s) to attempt to claim ownership for UNASSIGNED staged jobs (Job Claimer Service), job execution, and job-level metrics reporting (Metrics Reporter service). When this property is disabled, the Redis Connect instance may still participate in cluster leader election, facilitate REST API / CLI, and perform Job Reaper services.	true
job.claim.attempt.interval	Integer	Interval at which this scheduled thread attempts to claim ownership for UNASSIGNED staged jobs. Runs on each Redis Connect Instance (JVM) when `job.claim.service.enabled=true`. Measured in milliseconds with a minimum of 1 second (1000 ms)	5000
job.claim.batch.size.per.attempt	Long	Specifies how many jobs can be claimed per attempt interval. If a sparse topology across many Redis Connect instances is desired, then lowering this interval is recommended.	4
job.claim.max.capacity	Integer	Specifies the maximum number of jobs that a single Redis Connect instance can claim at any given time.	4
job.claim.heartbeat.lease.renewal.ttl	Integer	TTL (Time-to-Live) which is renewed upon each iteration of a fixed rate scheduled thread that shares its value. Measured in milliseconds with a minimum of 1 second (1000 ms)	10000

REST API Properties

Table 5. REST API properties

Property name	Type	Description	Default
rest.api.enabled	Boolean	Instantiates an embedded Spring Boot Application to host the REST API and CLI.	true
rest.api.port	Integer	Specifies the port used for the REST API (and SWAGGER) powered by an embedded Spring Boot Application. If you are running multiple Redis Connect instances on the same server, each instance will require a different port for its REST API.	8282

Job Manager Database Properties

Table 6. Job Manager Database Properties

Property name	Type	Description	Default
redis.connection.url	String	A Redis URI indicating which Redis server to use for job management. For the Redis URI spec, the Lettuce documentation.	n/a
redis.connection.insecure	Boolean	Passed to Lettuce's `RedisURI.verifyPeer`. If true then verifyMode=FULL. Otherwise, if false, then verifyMode=NONE. When peer verification is disabled, Lettuce uses Netty's InsecureTrustManagerFactory.INSTANCE as the trust manager factory. Its javadoc notes that it should never be used in production and that it is purely for testing purposes.	false
redis.connection.timeout.duration	Integer	The timeout is canceled upon command completion/cancellation. Measured in seconds.	1
redis.connection.auto.reconnect	Boolean	Determine whether the driver will attempt to automatically reconnect to Redis. When enabled, then on disconnect, the client will try to reconnect, activate the connection and re-issue any queued commands.	true

Property name	Type	Description	Default
redis.connection.suspend.reconnect. on.protocol.failure	Boolean	When set to `true`, reconnect will be suspended on protocol errors. The reconnect itself has two phases: Socket connection and protocol/connection activation. In case a connection timeout occurs, a connection reset, or host lookup fails, this does not affect the cancellation of commands. In contrast, where the protocol/connection activation fails due to SSL errors or PING before activating connection failure, queued commands are canceled.	true
redis.connection.sslEnabled	Boolean	Enables SSL for one-way or mutual authentication. If this flag is set to `false`, TrustStore and KeyStore will not be passed to the client.	false

Job Manager Credentials Properties

Table 7. Job Manager Credentials Properties

Property name	Type	Description	Default
truststore.file.path	String	File path of the Java TrustStore (containing certificates trusted by the client)	n/a
keystore.file.path	String	File path of the Java KeyStore, which stores private key entries, certificates with public keys, or any other secret keys used for various cryptographic purposes.	n/a
credentials.dir.path	String	The name of the directory containing the Redis Connect credentials file. This directory path must include a properties file named `redisconnect_credentials_jobmanager.properties` [1].	../config/samples/credentials
credentials.rotation.event.listener.enabled	Boolean	When set to `true`, a listener will be created on the `redisconnect_credentials_jobmanager.properties` file within the `credentials.dir.path` to rotate credentials when they change. This lets you rotate credentials without restarting your Redis Connect instance.	false

Property name	Type	Description	Default
credentials.rotation.event.listener.interval	Integer	When `credentials.rotation.event.listener.enabled` is set to `true`, this flag sets the frequency at which is scanned for changes. Measured in milliseconds with a minimum of 60 seconds (60000 ms)	60000

Email Alerting Properties

Table 8. Email Alerting Properties

Property name	Type	Description	Default
mail.alert.enabled	Boolean	Enables email alerts when any error forces a job to stop.	false
mail.smtp.host	String	Hostname of the outgoing mail server.	smtp.gmail.com
mail.smtp.port	Integer	Set the non-SSL port number of the outgoing mail server.	587
mail.smtp.start.tls.enable	Boolean	Set or disable STARTTLS encryption[2].	true
mail.smtp.start.tls.required	Boolean	Set or disable the required STARTTLS encryption.	false
mail.to	String	The email address to send alerts to. This email address will also be used as the personal name. Multiple recipients can be added by delimiting them with a comma.	n/a
mail.debug	Boolean	Set session debugging on or off.	false

[1] Redis Connect never caches or persists credentials. Therefore, on each connection with the source, target, or job manager database, the credentials are read from a file. This enhances security and allows for seamless credential rotations and integration with secret management frameworks such as HashiCorp Vault.

[2] StartTLS is an extension of the SMTP protocol that tells the email server that the email client wants to use a secure connection using TLS or SSL.

Job Configuration

This section describes the fields in job configuration payload.

Job properties

Table 9. Job properties

Property name	Type	Description	Default
jobName	String	Unique name for a job. This name is used to derive all other Redis metadata keys related to the job execution workflow. Note: the jobName property is submitted separately from the job configuration when saving a job. In other words, jobName does not appear in the job configuration payload. Note: jobName should not be confused with jobId. jobIds are created as part of a job claim. They add-on a namespace to the jobName to identify the jobType and partitionId (if jobType=PARTITIONED_STREAM) [1].	n/a
partitions	Integer	Indicates how many partitions to create during startJob process. This attribute is ONLY used to partition a job with jobType=PARTITIONED_STREAM. Not jobType=LOAD. `CAUTION: Once a job has started, and job claims are created, a job cannot be repartitioned without deleting all job claims and existing checkpoints. Please reach out to Support to assist with the migration of checkpoints to avoid undesired outcomes.`	1
maxPartitionsPerClusterMember	Integer	The number of job partitions that can be claimed, and executed, on the same Redis Connect instance (JVM). If the limit forces partitions to span more instances than are currently deployed, then the job will not be able to start nor migrate [2].	0

Property name	Type	Description	Default
pipeline	JSON Object	See Job Pipeline Properties	n/a
source	JSON Object	See Job Source Properties	Not Null

Job Pipeline Properties

Table 10. Job Pipeline Properties

Property name	Type	Description	Default
pipelineBufferSize	Integer	Redis Connect's pipeline is powered by the LMAX Disruptor library (High Performance Inter-Thread Messaging). Must be a power of 2, minimum 1024 [3]	4096
preprocessorName	String	Functional interface (Consumer) that can be run before changed-data events are transformed and published to the pipeline. This is currently not extendable by end users.	n/a
postprocessorName	String	Functional interface (Consumer) that can be run after changed-data events are transformed and published to the pipeline. This is currently not extendable by end users.	n/a
stages	Job Pipeline Stage[]	See Job Pipeline Stage Properties	

Job Pipeline Stage Properties

Table 11. Job Pipeline Stage Properties

Property name	Type	Description	Default
stageName	String	Unique name which is used as an exact match reference to a custom-built target sink or a user-defined custom stage.	n/a
index	Integer	Specifies the sequence in which the stages of the pipeline should be orchestrated. Begins with 1 and each subsequent index should increment by 1	n/a

Property name	Type	Description	Default
metricsEnabled	Boolean	When enabled, the target sink stage will report throughput and latency related metrics for persistence in RedisTimeSeries. This can subsequently be visualized in Grafana.	false
metricsRetentionInHours	Long	Maximum duration for metrics samples as compared to the highest reported timestamp before they expire. Measured in hours; minimum is 1.	4
checkpointStageIndicator	Boolean	Indicates which sink will be responsible for committing the checkpoint to the target database. This is typically performed by the last stage of the pipeline and, often times, it is the only stage in the pipeline. Job pipeline can only have a single stage with checkpointStage `Indicator=true`	false
checkpointTransactionsEnabled	Boolean	Although the producer's polling event loop enqueues changed-data events in batches, each event is processed individually through the pipeline. This is because Redis Connect updates the checkpoint at the changed-data event level and not the batch [4].	false
keyPrefix	String	Adds a prefix to the target Redis key before the tableName and composition of targetKey enabled columns.	
userDefinedType	String	To create a custom stage, a factory interface must be extended so that Redis Connect can have visibility to it from a class loading perspective. See section X.X.X. The interface will force the user to create a `getType()` method which returns a unique String to represent the custom factory. This property must exactly match that custom unique String so that Redis Connect can properly discover and handle it as a custom stage.	
database		See Database Configuration Configuration for all target database configuration.	

Property name	Type	Description	Default
checkpointDatabase		See Database Properties Checkpoint database configuration. This is required only if Redis is not the target destination (which is only supported when Splunk is the target).	

Job Source Properties

Table 12. Job Source Properties

Property name	Type	Description	Default
pollSourceInterval	Long	Fixed rate interval representing how long to pause the producer's polling event loop if no new change events were found in the batch. Measured in milliseconds; minimum is 5	50
batchSize	Integer	Maximum # of events to dequeue from the source-event-queue AND maximum # of events to query from the source transaction log/table/queue upon each interval of the producer's polling event loop. Minimum is 1.	500
sourceTransactionTimeSequenceEnabled	Boolean	When enabled, the source commit/transaction timestamp (and sequence# if the timestamp is the same) will be used to calculate latency metrics and passed along as metadata for Redis Streams sink(s).	false
slowConsumerMaxRetryAttempts	Integer	-1 = UNLIMITED 0 = DISABLED 1+ = MAX_ATTEMPTS Used as part of back-pressure support for the data pipeline in the event of a slow consumer. If the maximum attempts limit is reached, the job will be stopped for purposes of manual intervention.	50
intermittentEventSleepDuration	Integer	Used as part of back-pressure support for the data pipeline in the event of a slow consumer or the circuit breaker is open. Forces the event loop to pause for the configured duration of time. Measured in milliseconds, minimum is 0.	3

Property name	Type	Description	Default
sourceConnectionMaxRetryAttempts	Integer	0 = DISABLED 1+ = MAX_ATTEMPTS Maximum retry attempts to reconnect with the source in the event that a connection is lost. Minimum is 0.	3
sourceConnectionMaxRetryDuration	Integer	In addition to sourceConnectionMaxRetryAttempts, you can also add a max duration, after which retries will stop if the max attempts haven't already been reached. Measured in minutes; minimum is 1.	5
sourceConnectionRetryDelayInterval	Long	Fixed delay in between sourceConnectionMaxRetryAttempts. Measured in seconds; minimum is 0[5].	60
sourceConnectionRetryMaxDelayInterval	Long	Provides an upper bound to calculate the delay interval when sourceConnectionRetryDelayFactor is enabled. Measured in seconds, minimum is 0.	240
sourceConnectionRetryDelayFactor	Integer	0 = DISABLED 1+ = DELAY_FACTOR Factor by which delays are exponentially increased after each source connection retry attempt. Minimum is 0.	2
database	Object	See Database Configuration Configuration for all source databases. Not Null.	n/a
tables	Map<String, Table>	See Job Source Table Column Properties Configuration for all source tables/collections/regions/logs properties. Each table within the map requires a unique name which will be used as part of target key composition. Not Null.	n/a

Job Source Database Properties

See Database Configuration.

Job Source Table Properties

Table 13. Job Source Table Properties

Property name	Type	Description	Default
autoConfigColumnsEnabled	Boolean	When enabled, source metadata is queried during the (re)start process to determine sourceColumn names so users do not need to enumerate each within the column's configuration. The columns configuration can be used to override source metadata (i.e., targetName, type, etc.). However, targetKey designation cannot be overridden since only the source table's primary key will be used. This is a common configuration in POCs and development environments since the design of Redis key names are less important than in production. It also allows for less knowledge about the source table schema. This is only supported for RDBMS sources.	false
dynamicSchemaEnabled	Boolean	When enabled, columns that are not provided in the columns configuration will be passed through, as-is, to the target. This is currently only supported for MongoDB, Redis Streams Broker, and Files.	false
prefixTableNameToTargetKeyEnabled	Boolean	When enabled, adds the tableName (defined in the tables configuration) as a prefix to the target Redis key before all other targetKey enabled columns are computed and applied.	false

Property name	Type	Description	Default
deleteOnPrimaryKeyUpdateEnabled	Boolean	When enabled, if the primary key is changed at the source, then an additional operation to DELETE the existing target key will accompany the UPDATE event. This is only supported for RDBMS sources since primary key changes require a delete and insert of a new row. The DELETE event shares an offset with the UPDATE event both at the source and checkpoint. Redis Connect will handle them within a single pipeline iteration.	true
changedColumnsOnlyEnabled	Boolean	When enabled, only allows changed (delta) column values to be replicated to the target. This does not include targetKey column(s) which cannot be bypassed. When disabled, all column values will be replicated to the target unless they are individually bypassed at the column-level using changedColumnOnlyEnabled. (See Section 4.2.2) [6].	false
columns	Job Source Table Column[]	See Job Source Table Column Properties.	Null
initialLoad	Initial Load	See Job Source Table Initial Load Properties.	n/a

Job Source Table Column Properties

Table 14. Job Source Table Column Properties

Property name	Type	Description	Default
targetKey	Boolean	Designates this column's value as part of the target's key composition process. When more than one column is designated, the order in which they are listed will impact the order in which they are appended to the key.	false
sourceColumn	String	Exact match identifier for source column name. Must not be empty.	n/a
targetColumn	String	Preferred field name to be used in the target. Must not be empty.	n/a

Property name	Type	Description	Default
type	String	Identifies the source column's data type which is used to transform the column value to a properly formatted String within the target. Supported types include: [STRING, VARCHAR, TEXT, INT, DATE, DATE_TIME, BYTE, DEC, NUMERIC, DECIMAL, DOUBLE, FLOAT, LONG, SHORT, RAW, BLOB, CLOB, HASHMAP, CUSTOM] CUSTOM data type is unique in that it bypasses column value transformation to a String which allows it to be converted manually within a Custom Stage. An example would be converting to a proprietary Oracle Timestamp format. Failure to convert this data type manually will cause errors in Redis-based sinks.	STRING
changedColumnOnlyEnabled	Boolean	When enabled, only allows changed (delta) column values to be replicated to the target unless targetKey is enabled. When changedColumnsOnlyEnabled=true at the table-level, this flag will be overridden. This is currently only supported for RDBMS sources.	false
passThroughEnabled	Boolean	When disabled, the source column value will not be published to the pipeline therefore it cannot be accessed within a custom stage nor any sink. The purpose of this flag is to allow source column values to be used for targetKey composition without adding the column's name/value pair as a field within the target. As an example, this is common for sources like MongoDB which generate a "_id" key which can be used as a targetKey but has no value as a field.	true
index	Integer	This is currently for metadata purposes only and has no functional value.	n/a

Property name	Type	Description	Default
dateFormat	String	Used by DATE and DATE_TIME type to override their default. Default formats are as follows: DATE = YYYY-MM-dd DATE_TIME = YYYY-MM-dd HH:mm:ss.S	n/a
nullFormat	String	Users can define how a column value=NULL will be represented in the target.	Default is an EMPTY String.

Job Source Table Initial Load Properties

Table 15. Job Source Table Column Properties

Property name	Type	Description	Default
partitions	Integer	Indicates the number of partitions to create during startJob process. This attribute is ONLY used to partition an initial load with jobType=LOAD. Each table should be partitioned based on its own size and release window SLAs. It's common practice to leverage more partitions for an initial load than when streaming. Please see the Production Readiness section for more detail. Disclaimer: If the source table has fewer than 500 rows, which is common in a dev environment, all but partition:1 will be stopped so all the rows are loaded from a single partition. Minimum is 1.	1

Property name	Type	Description	Default
maxPartitionsPerClusterMember	Integer	Limits how many task partitions can be claimed, and executed, multi-tenant on the same Redis Connect instance (JVM). If the limit forces partitions to span more nodes than are currently deployed, then the initial load will queue the instantiation of tasks until capacity is reallocated (e.g. earlier tasks complete their load partition). This is not a job-level limit; it is only specific at the table level. 0 represents no limit. Minimum is 0.	0
customWhereClause	String	Users can specify a WHERE clause to filter the rows required for initial load. Only the following sources are supported: - RDBMS sources support JDBC compliant WHERE statements - MongoDB supports a BSON filter - Gemfire supports an Apache Geode WHERE Clause	
rowIndexUsedAsTargetKeyEnabled	Boolean	RDBMS sources can have tables without primary keys. For those cases, rowIndex can be used as a unique identifier for partitioning purposes. This is only supported for RDBMS sources and only for initial load only / ETL jobs.	false

[1] When jobName is used in logging or administrative processes (i.e., stopJob), the jobName represents ALL job partitions. Must be between 4 and 50 characters.

[2] For example, if maxPartitionsPerClusterMember=1 and partitions=3, then the Redis Connect cluster will require at least 3 instances (JVMs) each with at least 1 available capacity to claim a job partition. This is not a global limit; it is only specific at the job level 0 represents no limit

[3] The buffer size sets the number of slots allocated within the Disruptor's internal ring buffer "queue". Increasing the buffer size will impact the JVM heap space required to store all transient changed data events within the queue. For most cases, this can be left as default.

[4] When enabled, the checkpoint will be committed as part of an atomic Redis transaction. This eliminates consistency issues and improves performance. Rollback capability is built in to handle any failure scenarios during the transaction so that no data will be lost. When disabled, the checkpoint will be committed after the the changed-data events are written. This adds another network round trip for each changed-data event. While distributed checkpoints have distinct advantages, there is a small tradeoff. Because Redis keys are bound by their hash-slots, distributed checkpoints require that we store one checkpoint per hash slot. When enabled, each job partition will have its own 16384 checkpoint keys created in the target database. With ~250 bytes per checkpoint, each partition's estimated overhead is ~4MB. When disabled, there is only a single checkpoint key in the target per partition. Distributed checkpoints require RediSearch. We use RediSearch to index checkpoint keys so that recovery from the latest checkpoint is immediate

[5] sourceConnectionRetryDelayInterval must be < than sourceConnectionRetryDelayInterval

[6] When enabled, the column-level changedColumnOnlyEnabled flag will be overridden for all columns other than those designated as targetKey(s). This is currently only supported for RDBMS sources

Database Configuration

These properties are available for all **database** configuration objects. You will find database configuration objects in a job configuration's source and in the job configuration's pipeline, which specified the target database.

Table 16. Database Properties

Property name	Type	Description	Default
connectionType	String	Distinguishes between Job Manager, Job Source, Job Target, and Job Checkpoint databases. This field is auto-generated.	
databaseType	Enumeration	The following database types are supported: [DB2, FILES, GEMFIRE, MONGODB, ORACLE, POSTGRES, REDIS, REDIS_STREAMS_MESSAGE_BROKER, SPLUNK, SQL_SERVER, VERTICA] NONE is used for custom stages. Also see userDefinedType. This is a required field.	
databaseURL	String	Database URLs specify the subprotocol (the database connectivity mechanism), the database, or server identifier, and a list of properties. Not required for Gemfire, Splunk, and Files databaseTypes.	
credentialsDirectoryPath	String	Each database type other than NONE (used for custom jobs) requires a credentials property file, even if the credentials are not required. Credentials property files must adhere to the following filename pattern: redisconnect_credentials_{source_type | target_type}_{job_name}.properties The only exception is for Job Manager which has fixed name: `redisconnect_credentials_jobmanager.properties`	

Property name	Type	Description	Default
credentialsRotationEventListenerEnabled	Boolean	When enabled, the credentialsDirectoryPath will be periodically scanned for changes that are specific to the property file associated with this database[1].	false
credentialsRotationEventListenerInterval	Integer	Fixed rate scheduled interval that scans the credentialsDirectoryPath for changes when credentialsRotationEventListenerEnabled is enabled. Measured in milliseconds; minimum is 60000.	60000

[1] If a change is identified, the listener will create a new connection without bringing down the Redis Connect instance nor stopping the job. There might be a momentary pause in pipeline processing while the connection is being reestablished. No data will be lost in this process. Disclaimer: if the new credentials cannot be used to create a connection, the job will be stopped for manual intervention.

Custom Database Properties

RDBMS Properties

Table 17. Stream Job Type Custom Configurations

Property name	Type	Description	Default
column.exclude.list	List	An optional, comma-separated list of regular expressions that match the fully-qualified names of columns to exclude from change event record values. Fully-qualified names for columns are of the form `databaseName.tableName.columnName`. To match the name of a column, Debezium applies the regular expression that you specify as an anchored regular expression. That is, the specified expression is matched against the entire name string of the column; it does not match substrings that might be present in a column name. If you include this property in the configuration, do not also set the column.include.list property.	[]
decimal.handling.mode	String	Specifies how the connector should handle values for DECIMAL and NUMERIC types: “string” encodes values as formatted strings, which is easy to consume but semantic information about the real type is lost. “precise” represents them precisely using java.math.BigDecimal values represented in change events in a binary form. “double” represents them using double values, which may result in a loss of precision but is easier to use.	String
heartbeat.interval.ms	Long	Controls how frequently the connector sends heartbeat messages. Heartbeat messages are useful for ensuring the source transaction/redo logs and checkpoint are updated on a predictable basis. This can avoid a long period without updates which may cause a gap between the latest checkpoint and an archived transaction log that has been deleted. Default (0) does not send heartbeat messages. Measured in milliseconds; minimum is 0.	0

max.batch.size	Integer	Specifies the maximum size of each batch of events that should be processed during each iteration of the producer's polling event loop; minimum is 1.	2048
max.queue.size	Integer	Positive integer value that specifies the maximum number of records that the blocking queue can hold. When Debezium reads events streamed from the database, it places the events in the blocking queue before it writes them to Kafka. The blocking queue can provide backpressure for reading change events from the database in cases where the connector ingests messages faster than it can write them to Kafka, or when Kafka becomes unavailable. Events that are held in the queue are disregarded when the connector periodically records offsets. Always set the value of max.queue.size to be larger than the value of max.batch.size. `Note: max.queue.size > max.batch.size`	8192
snapshot.mode	String	Specifies the criteria for running a snapshot when the connector starts. It is not recommended to use this debezium capability for initial load in Production. See Production Deployment for more information. MySQL, Postgres default to 'never'; Oracle, SQL Server default to 'schema_only' [1].	n/a

Table 18. Initial Load Job Type Custom Configurations

Property name	Type	Description	Default
jdbc.connection.timeout	Long	Maximum duration the pool will wait for a connection to become available. Measured in milliseconds.	30000
jdbc.keepalive.time	Long	Fixed interval to keep a connection alive, in order to prevent it from being timed out by the database or network infrastructure. A "keepalive" will only occur on an idle connection. Default: 0 (disabled) Measured in milliseconds.	0

Property name	Type	Description	Default
jdbc.idle.timeout	Long	Takes effect only when the jdbc.min.idle is less than jdbc.max.pool.size, and is removed when the number of idle connections exceeds the jdbc.min.idle and the idle time exceeds jdbc.idle.timeout. If (jdbc.idle.timeout + 1 second) > jdbc.max.lifetime and jdbc.max.lifetime > 0, it will be reset to 0. If jdbc.idle.timeout != 0 and less than 10 seconds, it will be reset to 10 seconds. If jdbc.idle.timeout=0, idle connections will never be removed from the connection pool. Measured in milliseconds.	600000
jdbc.max.lifetime	Long	Maximum duration a connection is allowed to remain open. When this timeout is exceeded, the next time the connection is released to the pool it is closed, and a new connection is opened to replace it. Measured in milliseconds.	1800000
jdbc.max.pool.size	Integer	Maximum number of connections in the pool.	#partitions + 2
jdbc.min.idle	Integer	Controls the minimum number of connection pool idle connections. When the connection pool idle connections are less than jdbc.min.idle and the total number of connections is not more than jdbc.max.pool.size, it will try its best to supplement new connections.	#partitions
truststore.file.path	String	Holds onto certificates that identify 3rd parties. truststore.type JKS String For Java keystore file format, this property has the value jks (or JKS). You do not normally specify this property, because its default value is already jks.	n/a
keystore.file.path	String	Stores private key entries, certificates with public keys, or just secret keys that we may use for various cryptographic purposes. keystore.type JKS String Depending on what entries the keystore can store and how the keystore can store the entries, there are a few different types of keystores in Java: JKS, JCEKS, PKCS12, PKCS11 and DKS.	N/A

MongoDB Properties

Table 19. Stream Job Type Custom Configurations

Property name	Type	Description	Default
connect.backoff.initial.delay.ms	Integer	Initial delay when trying to reconnect to a primary after the first failed connection attempt or when no primary is available. Measured in milliseconds.	1000
connect.backoff.max.delay.ms	Integer	Maximum delay when trying to reconnect to a primary after repeated failed connection attempts or when no primary is available. Measured in milliseconds.	120000
connect.max.attempts	Integer	Maximum number of failed connection attempts to a replica set primary before an exception occurs and task is aborted.	3
mongodb.authsource	String	Database (authentication source) containing MongoDB credentials. This is required only when MongoDB is configured to use authentication with another authentication database than admin.	
admin.connection.pool.max.size	Integer	Maximum number of connections opened in the pool.	100
connection.pool.min.size	Integer	Minimum number of connections opened in the pool	0
connection.timeout	Integer	Maximum duration to establish a connection before an error is thrown. Depending on network infrastructure and load on the server, the client may have to wait for a connection establishment. Possible scenarios where connection timeout can happen – the server gets shut down, network issues, wrong IP/DNS, port number etc. Measure in milliseconds.	10000
mongodb.ssl.invalid.hostname.allowed	Boolean	When SSL is enabled, this setting controls whether strict hostname checking is disabled during connection phase. If true, the connection will not prevent man-in-the-middle attacks.	False
read.timeout	Integer	Maximum duration to send or receive on a socket before an error is thrown. “0” default indicates disabling the timeout. Measured in milliseconds."	0
mongodb.ssl.enabled	Boolean	Enables TLS/SSL connection.	False

Files Connector Properties

Table 20. Initial Load Job Type Custom Configurations

Property name	Type	Description	Default
charset	String	Defines a mapping between sequences of sixteen-bit UTF-16 code units (that is, sequences of chars) and sequences of bytes.	UTF-16
column.names	String	If a header row is not provided, column names can be configured manually. Order matters for proper association. Supported for delimited types. CSV, PSV, TSV	n/a
continuation.identifier	String	Any arbitrary String (special characters) to concatenate multiple sequential lines.	\\
delimiter	String	Sequence of one or more characters for specifying the boundary between separated column-values.	,
google.cloud.storage.project.id	String	Unique identifier for a project within the GCP console.	n/a
gzip.file.format	Boolean	gzip is a file format and a software application used for file compression and decompression.	False
header.line	Integer	Specifies which row is the header line since some delimited file types have comments or other non-column related headers.	0
included.fields	String	Whitelist of fields/columns that should be included for replication. All others will be ignored.	n/a
includes.header	Boolean	Most delimited file types include a header row which can be used to name each column.	True
initial.offset	Integer	Specifies row/offset from which all scanning should begin inclusive of the header line in delimited types.	0
source.file.path	String	Location of the file to be loaded.	n/a
quote.character	Char	Avoids syntax errors when double quotes are used within field values. Any character can be used as a substitute.	"
type	String	Supported file formats include: CSV, JSON, PSV, TSV, XML.	N/A

Gemfire Properties

Table 21. Custom Configurations

Property name	Type	Description	Default
function.filter.batch.size			
pool.free.connection.timeout	Integer	Number of milliseconds (ms) that the client waits for a free connection if max-connections limit is configured and all connections are in use.	10000
pool.idle.timeout	Integer	Number of milliseconds to wait for a connection to become idle for load balancing	5000
pool.locator.host			
pool.locator.port			
pool.load.conditioning.interval	Integer	Interval in which the pool checks to see if a connection to a specific server should be moved to a different server to improve the load balance.	300000 (5m)
pool.max.connections	Integer	Maximum number of connections that the pool can create. If all connections are in use, an operation requiring a client-to server-connection is blocked until a connection is available or the free-connection-timeout is reached. If set to -1, there is no maximum. The setting must indicate a cap greater than min-connections [2].	-1
pool.min.connections	Integer	Number of connections that must be created initially.	5
pool.multiuser.authentication	Boolean	Sets the pool to use multi-user secure mode. If in multiuser mode, then app needs to get RegionService instance of Cache	False
pool.ping.interval	Integer	Interval between pinging the server to show the client is alive, set in milliseconds. Pings are only sent when the ping-interval elapses between normal client messages. This must be set lower than the server's maximum-time-between-pings.	10000
pool.pr.single.hop.enabled	Boolean	Setting used for single-hop access to partitioned region data in the servers for some data operations. See PartitionResolver. See note in thread-local-connections below.	True

Property name	Type	Description	Default
pool.read.timeout	Integer	Number of milliseconds to wait for a response from a server before the connection times out.	10000
pool.retry.attempts	Integer	Number of times to retry an operation after a time-out or exception for high availability. If set to -1, the pool tries every available server once until it succeeds or has tried all servers.	-1
pool.server.group	String	Server group from which to select connections. If not specified, the global group of all connected servers is used.	n/a
pool.socket.buffer.size	Integer	Size of the socket buffer, in bytes, on each connection established.	32768
pool.socket.connect.timeout	Integer	Amount of time (in seconds) to wait for a response after a socket connection attempt.	59
pool.statistic.interval	Integer	Default frequency, in milliseconds, with which the client statistics are sent to the server. A value of -1 indicates that the statistics are not sent to the server.	-1
pool.subscription.ack.interval	Integer	Number of milliseconds to wait before sending an acknowledgment to the server about events received from the subscriptions.	100
pool.subscription.message.tracking.timeout	Integer	Number of milliseconds for which messages sent from a server to a client are tracked. The tracking is done to minimize duplicate events.	90000
pool.subscription.redundancy	Integer	Redundancy for servers that contain subscriptions established by the client. A value of -1 causes all available servers in the specified group to be made redundant.	0
pool.subscription.timeout.multiplier			
gf:queue:seq			

Redis Target Properties

Table 22. Event Handler Custom Configurations

Property name	Type	Description	Default
domain.strategy	Enumeration	Specifies the domain model that will be used to move data through the pipeline so that custom stages and targets are aligned. Domain Strategies include: DICTIONARY: passes all entities. KEY_ONLY: passes only key. STRING: passes key and single string blob. MESSAGE_BROKER: passes String[] with strict schema KEY_ONLY is used for invalidation and notification. STRING is used with REDIS_STRING_SINK. MESSAGE_BROKER is used for brokered deployments that stream data through Redis Streams. Only Splunk is currently supported for MESSAGE_BROKER.	n/a
databaseURL	String	See Lettuce's documentation for Redis URI syntax - https://lettuce.io/core/release/api/io/lettuce/core/RedisURI.html	N/A
redis.connection.insecure	Boolean	Passed to Lettuce's RedisURI.verifyPeer. If true then verifyMode=FULL, else if false then verifyMode=NONE. When peer verification is disabled, Lettuce uses Netty's InsecureTrustManagerFactory.INSTANCE as the trust manager factory. It's javadoc notes that it should never be used in production and that it is purely for testing purposes." FALSE redis.connection.timeout.duration Integer "Command timeout begins: When the command is sent successfully to the transport. Queued while the connection was inactive. The timeout is canceled upon command completion/cancellation. Measured in seconds.	1

Property name	Type	Description	Default
redis.connection.auto.reconnect	Boolean	Controls auto-reconnect behavior on connections. As soon as a connection gets closed/reset without the intention to close it, the client will try to reconnect, activate the connection and re-issue any queued commands. This flag also has the effect that disconnected connections will refuse commands and cancel these with an exception.	True
redis.connection.suspend.reconnect.on.protocol.failure	Boolean	If this flag is true the reconnect will be suspended on protocol errors. The reconnect itself has two phases: Socket connection and protocol/connection activation. In case a connection timeout occurs, a connection reset, host lookup fails, this does not affect the cancellation of commands. In contrast, where the protocol/connection activation fails due to SSL errors or PING before activating connection failure, queued commands are canceled.	True
redis.connection.sslEnabled	Boolean	Enables use of SSL for one-way or mutual authentication. If this flag is false, truststore and keystore will not be passed to the client.	False
redis.streams.max.length	Integer	Redis will trim the stream from the oldest entries when it reaches the number of entries specified in redis.streams.max.length. The stream could have more entries than specified in the command before Redis starts deleting entries.	0
redis.streams.passthrough.source.tx.time.enabled	Boolean	Redis Stream IDs are specified by two numbers separated by a - character. The first part is the Unix time in milliseconds of the Redis instance generating the ID. The second part is just a sequence number and is used in order to distinguish IDs generated in the same millisecond. When enabled, the source commit timestamp can be used to manually set the first part of the Redis Stream ID. Caution - time order is strictly enforced.	False

Property name	Type	Description	Default
redis.streams.sink.partitions	Integer	REDIS_STREAMS_SINK can partition the Redis Streams keys to which it will commit. This is complementary to job-level partitioning and does not replace it. If only the sink is partitioned a partition Id would be appended to the end of the Redis Streams key (e.g. key:2). If the sink is partitioned and the job was partitioned then first the job partition id would be appended to the Redis Streams key followed by the sink partition id (e.g. key:2:4).	0
redis.wait.enabled	Boolean	Blocks sink's commit from successfully completing until acknowledgment that the data was replicated to a backup Redis database shard. If checkpointTransactionsEnabled=true, the checkpoint will be rolledback to before this change occurred to avoid inconsistency and duplication. If checkpointTransactionsEnabled=false, the checkpoint won't be rolled back so on restart the operation would be duplicated.	False
redis.wait.timeout	Integer	If the timeout is reached, the sink's commit will fail and the consequence will be based on redis.wait.timeout.stop.job.enabled. Measured in milliseconds.	3000
redis.wait.timeout.stop.job.enabled	Boolean	If enabled, the job will be stopped and the quiesce process will be bypassed on the assumption that Redis is unavailable. If disabled, the error will be logged but the job will continue to operation. This introduces the risk that the data could be lost if the primary shard fails.	True

Redis Message Broker Properties

Table 23. Stream Job Type Custom Configurations

Property name	Type	Description	Default
redis.broker.eviction.strategy	Enumeration	If domain.strategy=MESSAGE_BROKER, Redis Streams is used as an event-stream(s) therefore requires cleanup to avoid running out of memory. To evict change-data events that are beyond the need for recovery, the following two options exist: THRESHOLD - evicts based on queue depth. SCHEDULER - evicts based on a scheduled thread.	THRESHOLD
redis.broker.eviction.scheduled.interval	Long	Fixed rate scheduled thread that evicts Redis Stream offsets that occurred before the oldest existing checkpoint. Measured in seconds.	10
redis.broker.eviction.threshold	Integer	Fixed queue depth threshold which triggers a separate thread to evict Redis Stream offsets that occurred before the oldest existing checkpoint. If redis.streams.eviction.strategy=THRESHOLD and redis.streams.eviction.threshold is not provided and job type is PARTITIONED_STREAM then redis.streams.eviction.threshold=redis.streams.max.length / partitions; If redis.streams.eviction.strategy=THRESHOLD and redis.streams.eviction.threshold is not provided and job type is STREAM then (redis.broker.max.queue.depth * 7) / 10;	n/a
redis.broker.max.queue.depth	Integer	Used to calculate redis.broker.eviction.threshold if it's not manually configured.	32768

Splunk Stream Properties

Table 24. Job Type Custom Configurations

Property name	Type	Description	Default
httpHeaders	String		
source.time.field.name	String		
source.time.sequence.field.name	String		

Property name	Type	Description	Default
authorization.stored.in.credentials.file	Boolean		
splunk.forwarder.destination.url	String		
http.post.max.retry.attempts	Integer		
http.post.max.retry.duration	Integer		
http.post.retry.delay.factor	Integer		
http.post.retry.delay.interval	Integer		
http.post.retry.max.delay.interval	Integer		

In-Memory Queue Properties

Table 25. Change Event Queue Custom Configurations

Property name	Type	Description	Default
poll.interval.ms	Long	Fixed rate interval that specifies the number of milliseconds the connector should wait for new changed-data events to appear before it starts processing a batch of events. Measured in milliseconds.	500
max.batch.size	Integer	Maximum events that can be dequeued by the source poll event loop within a single iteration.	16384
max.queue.size	Integer	Specifies the maximum number of records that the blocking queue can hold. [3].	32768
queue.persistence.enabled	Boolean	When enabled, batches of changed-data events are persisted to Redis Stream, before they are enqueued within the in-memory queue, which effectively mimics a change-data-capture (CDC) process within Redis Connect. [4].	True
queue.persistence.wait.enabled	Boolean	Blocks persistence from successfully completing until acknowledgment that the data was replicated to a backup Redis database shard completed. FALSE	queue.persistence.wait.timeout

[1] Possible settings are: initial - the connector runs a snapshot only when no offsets have been recorded for the logical server name. initial_only - the connector runs a snapshot only when no offsets have been recorded for the logical server name and then stops; i.e. it will not read change events from the binlog. when_needed - the connector runs a snapshot upon startup whenever it deems it necessary. That is, when no offsets are available, or when a previously recorded offset specifies a binlog location or GTID that is not available in the server. never - the connector never uses snapshots. Upon first startup with a logical server name, the connector reads from the beginning of the binlog. Configure this behavior with care. It is valid only when the binlog is guaranteed to contain the entire history of the database. schema_only - the connector runs a snapshot of the schemas and not the data. This

setting is useful when you do not need the topics to contain a consistent snapshot of the data but need them to have only the changes since the connector was started. schema_only_recovery - this is a recovery setting for a connector that has already been capturing changes. When you restart the connector, this setting enables recovery of a corrupted or lost database schema history topic. You might set it periodically to "clean up" a database schema history topic that has been growing unexpectedly. Database schema history topics require infinite retention

[2] If you use this setting to cap your pool connections, deactivate the pool attribute pr-single-hop-enabled. Leaving single hop activated can increase thrashing and lower performance.

[3] The blocking queue provides backpressure for reading changed data events from the source in cases where the connector ingests messages faster than they are consumed. Events that are held in the queue are disregarded when the connector periodically records offsets. Always set the value of maxQueueSize to be larger than the value of maxBatchSize

[4] Persistence allows for recovery of transient changed-data events which is critical for sources like Splunk that do not implement their own change-data-capture (CDC) process. Only supported by Gemfire and Splunk