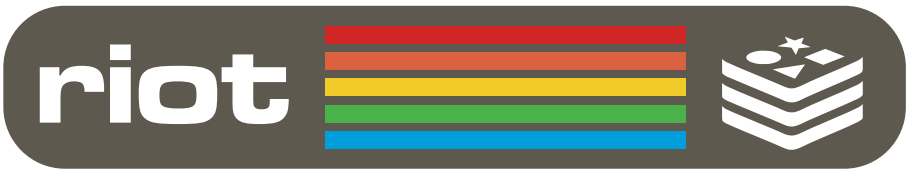



# Redis Input/Output Tools (RIOT)

Julien Ruaux

Version 3.2.0

# Table of Contents


Introduction . . . 1
Quick Start . . . 2
  Install . . . 2
  Usage . . . 2
Files . . . 4
  File Import . . . 4
  File Export . . . 8
  Dump Import . . . 10
Generators . . . 11
  Faker . . . 11
  Data Structures . . . 12
Databases . . . 13
  Database Import . . . 13
  Database Export . . . 14
  Database Drivers . . . 14
Replication . . . 16
  Usage . . . 16
  Replication Mode . . . 16
  Replication Type . . . 17
  Progress Reporting . . . 18
  Compare . . . 18
  Performance Tuning . . . 19
Architecture . . . 20
  Batching . . . 20
  Multi-threading . . . 20
  Processing . . . 21
  Filtering . . . 21
  Replication . . . 22
  Ping . . . 22
  Migrating from Elasticache . . . 22
Frequently Asked Questions . . . 26

# Introduction

Redis Input/Output Tools (RIOT) is a command-line utility to get data in and out of any Redis-compatible database like Redis OSS, Redis Cluster, Redis Enterprise, Redis Cloud, or Amazon ElastiCache.

RIOT includes the following features:

- File import/export (CSV, JSON, XML)
- Relational database import/export
- Data generation using Faker or random data structures
- Data migration from a Redis database to another
- Live replication between two Redis databases

# Quick Start

This section helps you get started with RIOT.

## Install

RIOT can be installed on Linux, macOS, and Windows platforms and can be used as a standalone tool that connects remotely to a Redis database. It is not required to run locally on a Redis server.

*Homebrew (macOS & Linux)*

```
brew install redis-developer/tap/riot
```

*Scoop (Windows)*

```
scoop bucket add redis-developer https://github.com/redis-developer/scoop.git
scoop install riot
```

*Manual installation (all platforms)*

Download the pre-compiled binary from the releases page, uncompress and copy to the desired location.

`riot-3.2.0.zip` requires Java 11 or greater to be installed. `riot-standalone-3.2.0-*.zip` includes its own Java runtime and does not require a Java installation.

*Docker*

```
docker run fieldengineering/riot [OPTIONS] [COMMAND]
```

## Usage

You can launch RIOT with the following command:

```
riot
```

This will show usage help, which you can also get by running:

```
riot --help
```

`--help` is available on any command and subcommand:

```
riot command --help
riot command subcommand --help
```

## Shell Completion

Run the following command to give `riot` TAB completion in the current shell:

```
$ source <(riot generate-completion)
```

## Manual

The manual is available here

# Files

RIOT can import from and export to files in various formats:

- Delimited (CSV, TSV, PSV)
- Fixed-length (also known as fixed-width)
- JSON and JSONL (JSON Lines)
- XML

## File Import

The `file-import` command reads data from files and writes it to Redis.

The basic usage for file imports is:

```
riot -h <host> -p <port> file-import FILE... [REDIS COMMAND...]
```

To show the full usage, run:

```
riot file-import --help
```

You must specify at least one Redis command as a target.

Redis connection options apply to the root command (`riot`) and not to subcommands.

In this example the Redis options will not be taken into account:

```
riot file-import my.json hset -h myredis.com -p 6380
```

The keys that will be written are constructed from input records by concatenating the keyspace prefix and key fields.


*Import into hashes with keyspace* `blah:<id>`

```
riot file-import my.json hset --keyspace blah --keys id
```

*Import into JSON*

```
riot file-import https://storage.googleapis.com/jrx/es_test-index.json json.set
--keyspace elastic --keys _id
```

*Import into hashes* **and** *set TTL on the key*

```
riot file-import my.json hset --keyspace blah --keys id expire --keyspace blah --keys
id
```

*Import into hashes in keyspace* `blah:<id>` **and** *set TTL* **and** *add each* `id` *to a set named* `myset`

```
riot file-import my.json hset --keyspace blah --keys id expire --keyspace blah --keys
id sadd --keyspace myset --members id
```

## Paths

Paths can include wildcard patterns.

RIOT will try to determine the file type from its extension (e.g. `.csv` or `.json`), but you can specify it with the `--filetype` option.

Gzipped files are supported and the extension before `.gz` is used (e.g. `myfile.json.gz` → JSON type).

*Examples*

- `/path/file.csv`
- `/path/file-*.csv`
- `/path/file.json`
- `http://data.com/file.csv`
- `http://data.com/file.json.gz`

Use `-` to read from standard input.

For AWS S3 buckets you can specify access and secret keys as well as the region for the bucket.

```
riot file-import s3://my-bucket/path/file.json --s3-region us-west-1 --s3-access
xxxxxx --s3-secret xxxxxx
```

For Google Cloud Storage you can specify credentials and project id for the bucket:

```
riot file-import gs://my-bucket/path/file.json --gcs-key-file key.json --gcs-project
-id my-gcp-project
```

## Delimited

The default delimiter character is comma (`,`). It can be changed with the `--delimiter` option.

If the file has a header, use the `--header` option to automatically extract field names. Otherwise specify the field names using the `--fields` option.

Let's consider this CSV file:

*Table 1. beers.csv*

| row | abv | ibu | id | name | style | brewery | ounces |
|---|---|---|---|---|---|---|---|
| 1 | 0.079 | 45 | 321 | Fireside Chat (2010) | Winter Warmer | 368 | 12.0 |
| 2 | 0.068 | 65 | 173 | Back in Black | American Black Ale | 368 | 12.0 |
| 3 | 0.083 | 35 | 11 | Monk's Blood | Belgian Dark Ale | 368 | 12.0 |

The following command imports this CSV into Redis as hashes using `beer` as the key prefix and `id` as primary key.

```
riot file-import https://storage.googleapis.com/jrx/beers.csv --header hset --keyspace
beer --keys id
```

This creates hashes with keys `beer:321`, `beer:173`, ...

This command imports a CSV file into a geo set named `airportgeo` with airport IDs as members:

```
riot file-import https://storage.googleapis.com/jrx/airports.csv --header --skip-limit
3 geoadd --keyspace airportgeo --members AirportID --lon Longitude --lat Latitude
```

## Fixed-Length

Fixed-length files can be imported by specifying the width of each field using the `--ranges` option.

```
riot file-import https://storage.googleapis.com/jrx/accounts.fw --ranges 1 9 25 41 53
67 83 --header hset --keyspace account --keys Account
```

## JSON

The expected format for JSON files is:

```
[
  {
    "...": "..."
  },
  {
    "...": "..."
  }
]
```

*JSON import example*

```
riot file-import https://storage.googleapis.com/jrx/beers.json hset --keyspace beer
--keys id
```

JSON records are trees with potentially nested values that need to be flattened when the target is a Redis hash for example.

To that end, RIOT uses a field naming convention to flatten JSON objects and arrays:

*Table 2. Nested object*

| | | |
|---|---|---|
| `{ "field": { "sub": "value" } }` | → | `field.sub=value` |

*Table 3. Array*

| | | |
|---|---|---|
| `{ "field": [1, 2, 3] }` | → | `field[0]=1 field[1]=2 field[2]=3` |

## XML

Here is a sample XML file that can be imported by RIOT:

```
<?xml version="1.0" encoding="UTF-8"?>
<records>
    <trade>
        <isin>XYZ0001</isin>
        <quantity>5</quantity>
        <price>11.39</price>
        <customer>Customer1</customer>
    </trade>
    <trade>
        <isin>XYZ0002</isin>
        <quantity>2</quantity>
        <price>72.99</price>
        <customer>Customer2c</customer>
    </trade>
    <trade>
        <isin>XYZ0003</isin>
        <quantity>9</quantity>
        <price>99.99</price>
        <customer>Customer3</customer>
    </trade>
</records>
```

*XML Import Example*

```
riot file-import https://storage.googleapis.com/jrx/trades.xml hset --keyspace trade
--keys id
```

### Regular Expressions

In addition to general processing you can perform field extraction using regular expressions.

*Regex example*

```
riot file-import --regex name="(?<first>\w+)\/(?<last>\w+)" ...
```

# File Export

The `file-export` command reads data from a Redis database and writes it to a JSON or XML file, potentially gzip-compressed.

The general usage is:

```
riot -h <host> -p <port> file-export FILE
```

To show the full usage, run:

```
riot file-export --help
```

*Export to JSON*

```
riot file-export /tmp/redis.json
```

*Sample JSON-export file*

```
[
  {
    "key": "string:615",
    "ttl": -1,
    "value": "value:615",
    "type": "STRING"
  },
  {
    "key": "hash:511",
    "ttl": -1,
    "value": {
      "field1": "value511",
      "field2": "value511"
    },
    "type": "HASH"
  },
  {
    "key": "list:1",
    "ttl": -1,
    "value": [
      "member:991",
      "member:981"
    ],
    "type": "LIST"
  },
  {
    "key": "set:2",
    "ttl": -1,
    "value": [
      "member:2",
      "member:3"
    ],
    "type": "SET"
  },
  {
    "key": "zset:0",
    "ttl": -1,
    "value": [
      {
        "value": "member:1",
        "score": 1.0
```

```
      }
    ],
    "type": "ZSET"
  },
  {
    "key": "stream:0",
    "ttl": -1,
    "value": [
      {
        "stream": "stream:0",
        "id": "1602190921109-0",
        "body": {
          "field1": "value0",
          "field2": "value0"
        }
      }
    ],
    "type": "STREAM"
  }
]
```


*Export to compressed JSON*

```
riot file-export /tmp/beers.json.gz --scan-match beer:*
```

*Export to XML*

```
riot file-export /tmp/redis.xml
```

# Dump Import

RIOT can import Redis data structure files in JSON or XML formats. See File Export section to generate such files.

*Example*

```
riot dump-import /tmp/redis.json
```

# Generators

RIOT includes two data generators that can be used to quickly mock up a dataset in Redis.

## Faker

The `faker-import` command generates data using Datafaker.

```
riot -h <host> -p <port> faker-import SPEL... [REDIS COMMAND...]
```

where SPEL is a Spring Expression Language field in the form `field="expression"`.

To show the full usage, run:

```
riot faker-import --help
```

You must specify at least one Redis command as a target.

Redis connection options apply to the root command (`riot`) and not to subcommands.

In this example the Redis options will not be taken into account:

```
riot faker-import id="index" hset -h myredis.com -p 6380
```

### Keys

Keys are constructed from input records by concatenating the keyspace prefix and key fields.


*HSET example*

```
riot faker-import id="index" firstName="name.firstName" lastName="name.lastName"
address="address.fullAddress" hset --keyspace person --keys id
```

*SADD example*

```
riot faker-import name="gameOfThrones.character" --count 1000 sadd --keyspace
got:characters --members name
```

### Data Providers

Faker offers many data providers. Most providers don't take any arguments and can be called directly:

*Simple Faker example*

```
riot faker-import firstName="name.firstName"
```

Some providers take parameters:

*Parameter Faker example*

```
riot faker-import lease="number.digits(2)"
```

Refer to Datafaker Providers for complete documentation.

### Built-in Fields

In addition to the Faker fields specified with `field="expression"` you can use these built-in fields:

`index`
: current iteration number.

`thread`
: current thread id. Useful for multithreaded data generation.

*Multithreaded data generator*

```
riot faker-import --count 8000 --threads 8 id="thread*10000+index"
firstName="name.firstName" lastName="name.lastName" address="address.fullAddress" hset
--keyspace person --keys id
```

### RediSearch

You can infer Faker fields from a RediSearch index using the `--infer` option:

```
riot faker-import --infer beerIdx hset --keyspace beer --keys id
```

## Data Structures

The `generate` command generates data-structures for Redis, RedisJSON and RedisTimeSeries.

```
riot -h <host> -p <port> generate [OPTIONS]
```

# Databases

RIOT can import from and export to databases.

## Database Import

The `db-import` command imports data from a relational database into Redis.

Ensure RIOT has the relevant JDBC driver for your database. See the Drivers section for more details.

```
riot -h <redis host> -p <redis port> db-import --url <jdbc url> SQL [REDIS COMMAND...]
```

To show the full usage, run:

```
riot db-import --help
```

You must specify at least one Redis command as a target.

Redis connection options apply to the root command (`riot`) and not to subcommands.

In this example the Redis options will not be taken into account:

```
riot db-import "SELECT * FROM customers" hset -h myredis.com -p 6380
```

The keys that will be written are constructed from input records by concatenating the keyspace prefix and key fields.


*PostgreSQL Import Example*

```
riot db-import "SELECT * FROM orders" --url "jdbc:postgresql://host:port/database"
--username appuser --password passwd hset --keyspace order --keys order_id
```

*Import from PostgreSQL to JSON strings*

```
riot db-import "SELECT * FROM orders" --url "jdbc:postgresql://host:port/database"
--username appuser --password passwd set --keyspace order --keys order_id
```

This will produce Redis strings that look like this:

```
{
  "order_id": 10248,
  "customer_id": "VINET",
  "employee_id": 5,
  "order_date": "1996-07-04",
  "required_date": "1996-08-01",
  "shipped_date": "1996-07-16",
  "ship_via": 3,
  "freight": 32.38,
  "ship_name": "Vins et alcools Chevalier",
  "ship_address": "59 rue de l'Abbaye",
  "ship_city": "Reims",
  "ship_postal_code": "51100",
  "ship_country": "France"
}
```

# Database Export

Use the `db-export` command to read from a Redis database and writes to a SQL database.

The general usage is:

```
riot -h <redis host> -p <redis port> db-export --url <jdbc url> SQL
```

To show the full usage, run:

```
riot db-export --help
```

*Example: export to PostgreSQL*

```
riot db-export "INSERT INTO mytable (id, field1, field2) VALUES (CAST(:id AS
SMALLINT), :field1, :field2)" --url "jdbc:postgresql://host:port/database" --username
appuser --password passwd --scan-match "gen:*" --key-regex "gen:(?<id>.*)"
```

# Database Drivers

RIOT relies on JDBC to interact with databases. It includes JDBC drivers for the most common database systems:

- Oracle

  `jdbc:oracle:thin:@myhost:1521:orcl`

- MS SQL Server

`jdbc:sqlserver://[serverName[\instanceName][:portNumber]][;property=value[;property=value]]`

- MySQL

  `jdbc:mysql://[host]:[port][/database][?properties]`

- PostgreSQL

  `jdbc:postgresql://host:port/database`

For non-included databases you must install the corresponding JDBC driver under the `lib` directory and modify the `CLASSPATH`:

- *nix: `bin/riot` → `CLASSPATH=$APP_HOME/lib/myjdbc.jar:$APP_HOME/lib/⋯`
- Windows: `bin\riot.bat` → `set CLASSPATH=%APP_HOME%\lib\myjdbc.jar;%APP_HOME%\lib...`

# Replication

Most Redis migration tools available today are offline in nature. Migrating data from AWS ElastiCache to Redis Enterprise Cloud for example means backing up your Elasticache data to an AWS S3 bucket and importing it into Redis Enterprise Cloud using its UI.

Redis has a replication command called REPLICAOF but it is not always available (see ElastiCache restrictions). Instead, RIOT implements client-side replication using **dump & restore** or **type-based read & write**. Both snapshot and live replication modes are supported.

Please note that RIOT is NEITHER recommended NOR officially supported by Redis, Inc.

## Usage

```
riot <source> replicate <target> --mode <snapshot|live|compare> [--type] [OPTIONS]
```

For the full usage, run:

```
riot replicate --help
```

## Replication Mode

### Snapshot

This mode iterates over the keys in the source Redis database using scan.

### Live

This mode builds upon snapshot replication by listening for changes on the source Redis database. Whenever a key is modified its corresponding value is read and propagated to the target Redis database.

Live replication relies on keyspace notifications for capturing these changes.

**Make sure the source database has keyspace notifications enabled** using:

- `redis.conf`: `notify-keyspace-events = KA`
- `CONFIG SET notify-keyspace-events KA`

For more details see Redis Keyspace Notifications.

The live replication mechanism does not guarantee data consistency. Redis sends keyspace notifications over pub/sub which does not provide guaranteed delivery. It is possible that RIOT can miss some notifications in case of network failures for example.

Also, depending on the type, size, and rate of change of data structures on the source it is possible that RIOT cannot keep up with the change stream. For example if a big set is repeatedly updated, RIOT will need to read the whole set on each update and transfer it over to the target database. With a big-enough set, RIOT could fall behind and the internal queue could fill up leading up to updates being dropped.

For those potentially problematic migrations it is recommend to perform some preliminary sizing using Redis statistics and `bigkeys`/`memkeys` in tandem with `--mem-limit`. If you need assistance please contact your Redis account team.

# Replication Type

## Dump & Restore

The default replication mechanism is Dump & Restore:

1. Scan for keys in the source Redis database. If live replication is enabled the reader also subscribes to keyspace notifications to generate a continuous stream of keys.
2. Reader threads iterate over the keys to read corresponding values (DUMP) and TTLs.
3. Reader threads enqueue key/value/TTL tuples into the reader queue, from which the writer dequeues key/value/TTL tuples and writes them to the target Redis database by calling RESTORE and EXPIRE.

## Type-Based Replication

There are situations where Dump & Restore cannot be used, for example:

- The target Redis database does not support the RESTORE command (Redis Enterprise CRDB)
- Incompatible DUMP formats between source and target (Redis 7.0)

In those cases you can use another replication strategy called **Type-Based Replication** where each key is introspected to determine the type of data structure and which read/write commands to use:

| Type | Read | Write |
|---|---|---|
| **Hash** | `HGETALL` | `HSET` |
| **JSON** | `JSON.GET` | `JSON.SET` |
| **List** | `LRANGE` | `RPUSH` |
| **Set** | `SMEMBERS` | `SADD` |
| **Sorted Set** | `ZRANGE` | `ZADD` |

| Type | Read | Write |
|---|---|---|
| **Stream** | `XRANGE` | `XADD` |
| **String** | `GET` | `SET` |
| **TimeSeries** | `TS.RANGE` | `TS.ADD` |

This replication strategy is more intensive in terms of CPU, memory, and network for the machines running RIOT as well as the source and target Redis databases. Adjust number of threads, batch and queue sizes accordingly.

*Snapshot replication example*

```
riot -h source -p 6379 replicate -h target -p 6380 --batch 10
```

*Live replication example*

```
riot -h source -p 6379 replicate -h target -p 6380 --mode live
```

*Type-based replication example*

```
riot -h source -p 6379 replicate -h target -p 6380 --type
```

*Live type-based replication example*

```
riot -h source -p 6379 replicate -h target -p 6380 --type --mode live --compare none
```

# Progress Reporting

Each process (scan iterator and/or event listener in case of live replication) has a corresponding status bar that shows the process name and its progress:

**Scanning**

Percentage of keys that have been replicated ⇒ replicated / total. The total number of keys is calculated when the process starts and it can change by the time it is finished (for example if keys are deleted or added during the replication). The progress bar is only a rough indicator.

**Listening**

Progress is indefinite as total number of keys is unknown.

# Compare

Once replication is complete, RIOT performs a verification step by iterating over keys in the source database and comparing values and TTLs between source and target databases.

The verification step happens automatically after the scan is complete (snapshot replication), or for

live replication when keyspace notifications have become idle (see Usage section).

Verification can also be run on-demand using the `compare` mode:

```
riot <source> replicate --mode compare <target>
```

The output looks like this:

```
missing: 123, type: 54, value: 7, TTL: 19
```

**missing**
: Number of keys only present in source database

**type**
: Number of keys with mismatched data structure type

**value**
: Number of keys with mismatched value

**TTL**
: Number of keys with mismatched TTL i.e. difference is greater than tolerance (can be specified with `--ttl-tolerance`)

There are 2 comparison modes available through `--compare-mode`:

- Quick (default): compares key types
- Full: compares key types, values, and TTLs

To show which keys are different use the `--show-diffs` option:

```
riot <source> replicate <target> --show-diffs
```

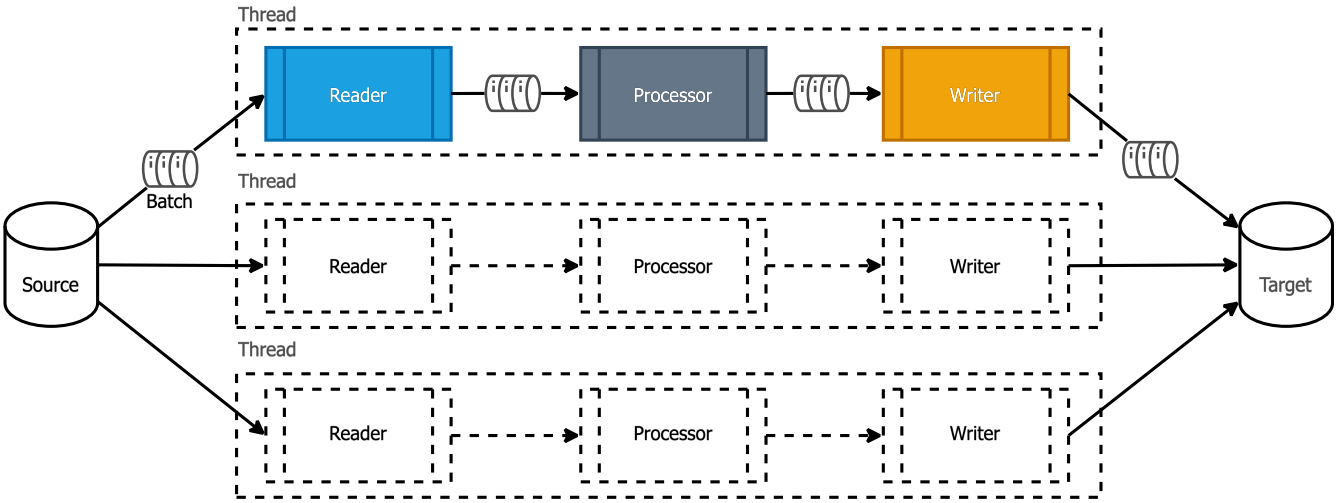
# Performance Tuning

Performance tuning is an art but RIOT offers some options to identify potential bottlenecks. In addition to batch and threads options you have the `--dry-run` option which disables writing to the target Redis database so that you can tune the reader in isolation. Add that option to your existing `replicate` command-line to compare replication speeds with and without writing to the target Redis database:

```
riot <source> replicate <target> --dry-run
```

# Architecture

RIOT is essentially an ETL tool where data is extracted from the source system, transformed (see Processing), and loaded into the target system.




## Batching

Processing in RIOT is done in batches: a fixed number of records is read from the source, processed, and written to the target. The default batch size is `50`, which means that an execution step reads 50 items at a time from the source, processes them, and finally writes then to the target. If the target is Redis, writing is done in a single command (Redis Pipelining) to minimize the number of roundtrips to the server.

You can change the batch size (and hence pipeline size) using the `--batch` option. The optimal batch size in terms of throughput depends on many factors like record size and command types (see Redis Pipeline Tuning for details).

## Multi-threading

It is possible to parallelize processing by using multiple threads. In that configuration, each chunk of items is read, processed, and written in a separate thread of execution. This is different from partitioning where items would be read by multiple readers. Here, only one reader is being accessed from multiple threads.

To set the number of threads, use the `--threads` option.

*Multi-threading example*

```
riot db-import "SELECT * FROM orders" --url "jdbc:postgresql://host:port/database"
--username appuser --password passwd --threads 3 hset --keyspace order --keys order_id
```

# Processing

RIOT lets you transform incoming records using processors. These processors allow you to create/update/delete fields using the Spring Expression Language (SpEL). For example, import commands like `file-import`, `database-import`, and `faker-import` have a `--proc` option that allow for field-level processing:

- `field1='foo'` → generate a field named `field1` containing the string `foo`
- `temp=(temp-32)*5/9` → convert from Fahrenheit to Celsius
- `name=remove(first).concat(remove(last))` → concatenate `first` and `last` fields and delete them
- `field2=null` → delete `field2`

Input fields are accessed by name (e.g. `field3=field1+field2`).

Processors have access to the following context variables and functions:

`date`
: Date parsing and formatting object. Instance of Java SimpleDateFormat.

`redis`
: Redis commands object. Instance of Lettuce RedisCommands.

`geo`
: Convenience function that takes a longitude and a latitude to produce a RediSearch geo-location string in the form `longitude,latitude` (e.g. `location=#geo(lon,lat)`)

*Processor example*

```
riot file-import --proc epoch="#date.parse(mydate).getTime()" location="#geo(lon,lat)"
name="#redis.hget('person1','lastName')" ...
```

You can register your own variables using `--var`.

*Custom variable example*

```
riot file-import https://storage.googleapis.com/jrx/lacity.csv --var rnd="new
java.util.Random()" --proc randomInt="#rnd.nextInt(100)" --header hset --keyspace
event --keys Id
```

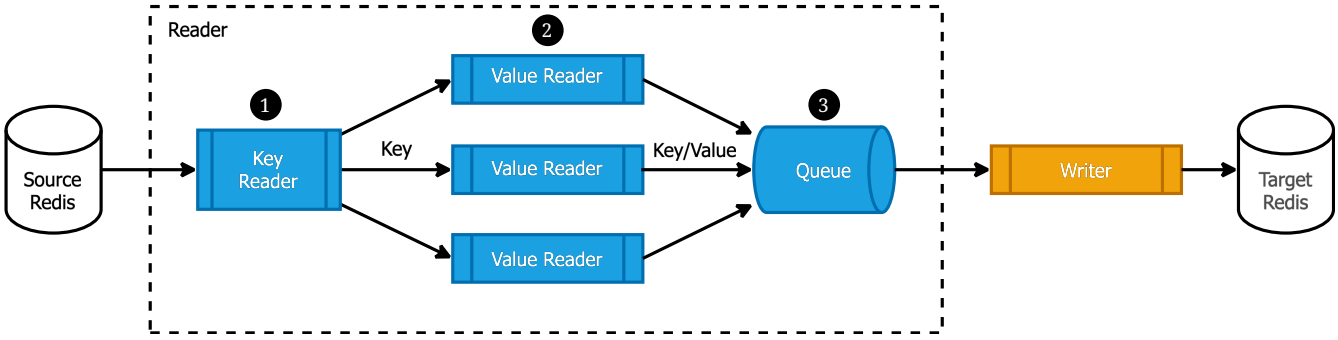
# Filtering

Filters allow you to exclude records that don't match a SpEL boolean expression.

For example this filter will only keep records where the `value` field is a series of digits:

```
riot file-import --filter "value matches '\\d+'" ...
```

# Replication




The basic replication mechanism is as follows:

1. Identify source keys to be replicated using scan and/or keyspace notifications depending on the replication mode.
2. Read data associated with each key using dump or type-specific commands.
3. Write each key to the target using restore or type-specific commands. = Cookbook

Here are various recipes using RIOT.

# Ping

The `ping` command can be used to test connectivity to a Redis database.

```
riot -h <host> -p <port> ping <options>
```

When the command is complete you will see statistics like these:

```
[min=0, max=19, percentiles={50.0=1, 90.0=3, 95.0=6, 99.0=10, 99.9=17}]
```

# Migrating from Elasticache

This recipe contains step-by-step instructions to migrate an Elasticache (EC) database to Redis Enterprise (RE).

The following scenarios are covered:

- One-time (snapshot) migration
- Online (live) migration

It is recommended to read the Replication section to familiarize yourself with its usage and architecture.

## Setup

### Prerequisites

For this recipe you will require the following resources:

- AWS ElastiCache: *Primary Endpoint* in case of Single Master and *Configuration Endpoint* in case of Clustered EC. Refer to this link to learn more
- Redis Enterprise: hosted on Cloud or On-Prem
- An Amazon EC2 instance

*Keyspace Notifications*

For a live migration you need to enable keyspace notifications on your Elasticache instance (see AWS Knowledge Center).

### Migration Host

To run the migration tool we will need an EC2 instance.

You can either create a new EC2 instance or leverage an existing one if available. In the example below we first create an instance on AWS Cloud Platform. The most common scenario is to access an ElastiCache cluster from an Amazon EC2 instance in the same Amazon Virtual Private Cloud (Amazon VPC). We have used Ubuntu 16.04 LTS for this setup but you can choose any Ubuntu or Debian distribution of your choice.

SSH to this EC2 instance from your laptop:

```
ssh -i "public key" <AWS EC2 Instance>
```

Install `redis-cli` on this new instance by running this command:

```
sudo apt update
sudo apt install -y redis-tools
```

Use `redis-cli` to check connectivity with the Elasticache database:

```
redis-cli -h <ec primary endpoint> -p 6379
```

Ensure that the above command allows you to connect to the remote Elasticache database successfully.

### Installing RIOT

Let's install RIOT on the EC2 instance we set up previously. For this we'll follow the steps in Manual Installation.

## Performing Migration

We are now all set to begin the migration process. The options you will use depend on your source and target databases, as well as the replication mode (snapshot or live).

### EC Single Master → RE

```
riot -h <source EC host> -p <source EC port> replicate -h <target RE host> -p <target
RE port> --pass <RE password>
```

### Live EC Single Master → RE

```
riot -h <source EC host> -p <source EC port> replicate --mode live -h <target RE host>
-p <target RE port> --pass <RE password>
```

In case ElastiCache is configured with AUTH TOKEN enabled, you need to pass `--tls` as well as `--pass` option:

```
riot -h <source EC host> -p <source EC port> --tls --pass <token>
replicate -h <target RE host> -p <target RE port> --pass <RE password>
```

### EC Cluster → RE

```
riot -h <source EC host> -p <source EC port> --cluster replicate -h <target RE host>
-p <target RE port> --pass <RE password>
```

`--cluster` is an important parameter used ONLY for ElastiCache whenever cluster-mode is enabled. Do note that the source database is specified first and the target database is specified after the replicate command and it is applicable for all the scenarios.

### EC Single Master → RE (with specific db index)

```
riot -h <source EC host> -p <source EC port> --db <index> replicate -h <target RE
host> -p <target RE port> --pass <RE password>
```

### EC Single Master → RE with OSS Cluster

```
riot -h <source EC host> -p <source EC port> replicate -h <target RE host> -p <target
RE port> --pass <RE password> --cluster
```

**Live EC Cluster → RE with OSS Cluster**

```
riot -h <source EC host> -p <source EC port> --cluster replicate --mode live -h
<target RE host> -p <target RE port> --pass <RE password> --cluster
```

## Important Considerations

- As stated earlier, this tool is not officially supported by Redis Inc.
- It is recommended to test migration in UAT before production use.
- Once migration is completed, ensure that application traffic gets redirected to Redis Enterprise Endpoint successfully.
- It is recommended to perform the migration process during low traffic so as to avoid chances of data loss.

# Frequently Asked Questions

**Logs are cut off or missing**

This could be due to concurrency issues in the terminal when refreshing the progress bar and displaying logs. Try running with job options `--progress log`.

**Unknown options: '--keyspace', '--keys'**

You must specify one or more Redis commands with import commands (`file-import`, `faker-import`, `db-import`).

**ERR DUMP payload version or checksum are wrong**

Redis 7 DUMP format is not backwards compatible with previous versions. To replicate between different Redis versions, use Type-Based Replication.

**Process gets stuck during replication and eventually times out**

This could be due to big keys clogging the replication pipes. In these cases it might be hard to catch the offending key(s). Try running the same command with `--info` and `--progress log` so that all errors are reported. Check the database with `redis-cli Big keys` and/or use reader options to filter these keys out.

**NOAUTH Authentication required**

This issue occurs when you fail to supply the `--pass <password>` parameter.