Solving the cold cache problem with Redis Data Integration (RDI)

Ricardo Ferreira
Lead Developer Advocate, DevRel

● Ricardo Ferreira

LEAD, DEVELOPER RELATIONS

→ Leads the developer relations team at Redis

→ Past DevRel experience: AWS, Elastic, and Confluent

→ 25+ years as a software engineer with strong focus on Java, but also occasionally writing code in Golang and Python

→ Areas of expertise: distributed systems, databases, in-memory data, data streaming, and observability

→ Feel free to add me on LinkedIn: **@riferrei**

What is the cold cache problem?

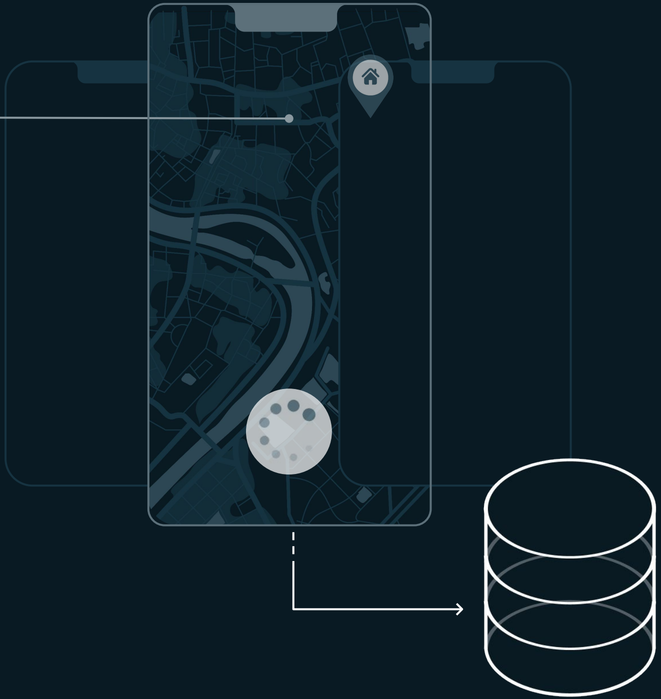
It all started with databases slowing down.

Limiting your ability to scale your workloads

→ Disk and network I/O getting in the way of transactions

→ Relying on database read replicas was expensive and troublesome

→ Using complex synchronization strategies developers dislike



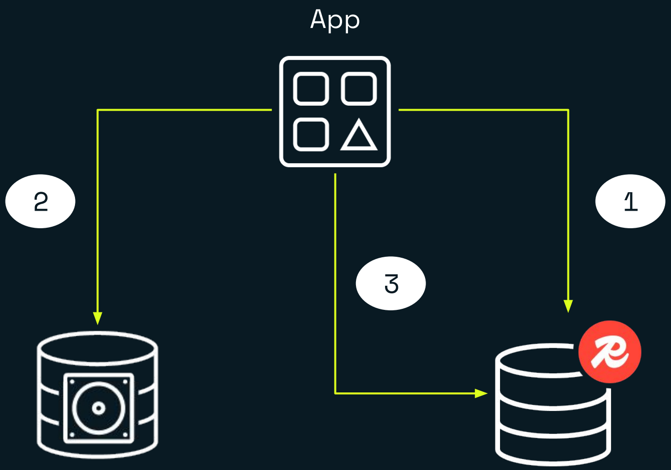
Solution: Using the cache aside pattern

Reads goes through Redis to offload your transactional database

1. App tries to read data from Redis. If there is cache hit, then data is returned immediately. This is the happy path.

2. If there is a cache miss, then read the data from the source database. This is what we want to avoid most of the time.

3. Update the cache with the recently read key, making it available for future calls.



But cache aside is not bulletproof.

Three main reasons why cache aside is not the perfect solution

Repetitive update logic

→ Each app must implement the cache aside pattern all over

→ It makes it hard to keep track of best practices across projects

→ Database changes break every new release of the application

Thundering herd problem

→ Many requests hit the database when the cache expires keys

→ Database must be sized to handle exporadic read loads

→ Query times start to get slower and slower, eventually causing errors

Data invalidation issues

→ What happens if records from the database are deleted?

→ How to pragmatically timeout expired keys?

→ There is no atomic way to write to both Redis and the database

Hey, it could be worse.

What is the cold cache problem?

A ”cold” cache is an empty cache with no data in it

Why it’s so important

→ First 1000 requests? All go straight to your database

→ Your database usually handles 20% load, now will handle 100%

→ Chances are your app will go offline in a matter of minutes

It usually happens when

→ Your app just started when requests are coming in

→ Perhaps you deployed a new version of the app in production

→ You manually flushed the cache or the keys expired naturally

Cascading failure pattern

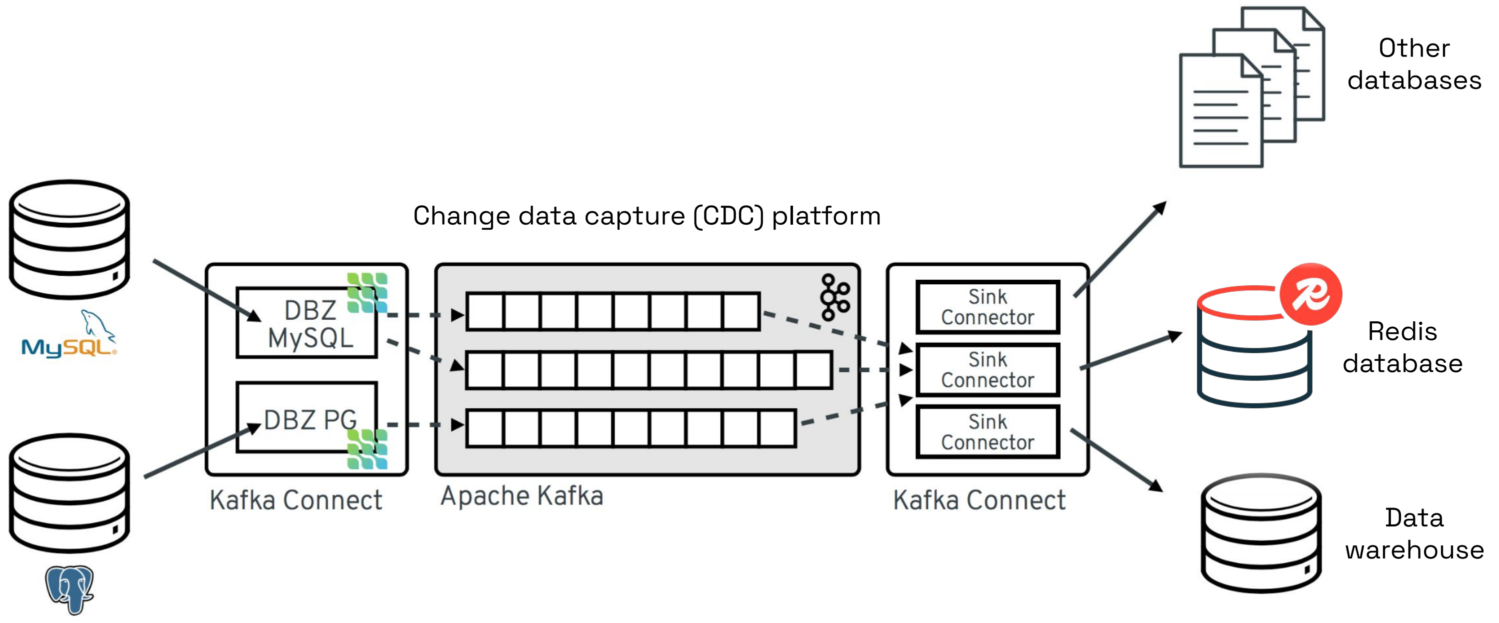
→ The database will slow down, causing queries to timeout

→ Cache can’t be updated since the database is timing out

→ System never recovers and manual intervention is required

How do we solve this?

Stream captured database events into Redis



Using CDC to stream data between your database & Redis adds complexity & cost.

Developer overutilization

Time spent building a new data pipeline to capture and transform changed data

Distributed systems hole

Kafka, Kafka Connect, and Debezium are very complex distributed systems

High costs

Excessive costs of building a solution with hard-to-hire experts in Apache Kafka

It makes you think twice before trying to solve your cold cache problem with CDC

You need a solution that doesn't hold you back.

Developer overutilization

Time spent building a new data pipeline to capture and transform changed data

✓

Developers fearlessly working on innovation

Keep your data pipelines working even when changes to your database are made

Distributed systems hole

Kafka, Kafka Connect, and Debezium are very complex distributed systems

✓

Operational simplicity for easy deployment of CDC

Work with an architecture that follows principles of simplicity for deployment

High costs

Excessive costs of building a solution with hard-to-hire experts in Apache Kafka

✓

Avoid excessive costs with an easy to onboard stack

Use an out-of-the box data pipeline to offload reads to Redis—speeding up the learning curve of adoption

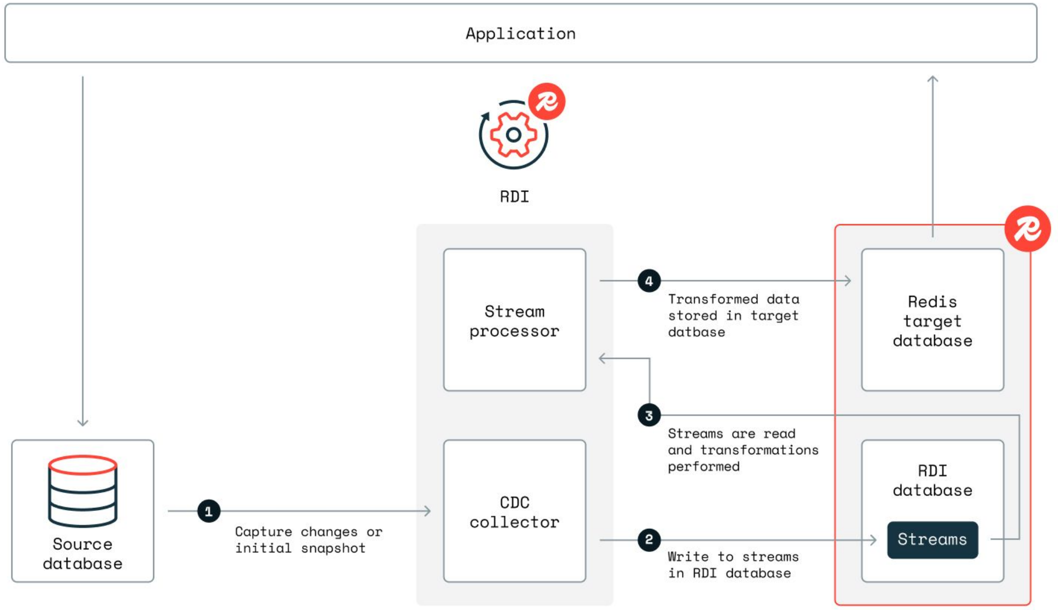
Meet Redis Data Integration (RDI).

Seamlessly sync Redis with your database

→ Easily configure and deploy a data pipeline between Redis and your databases

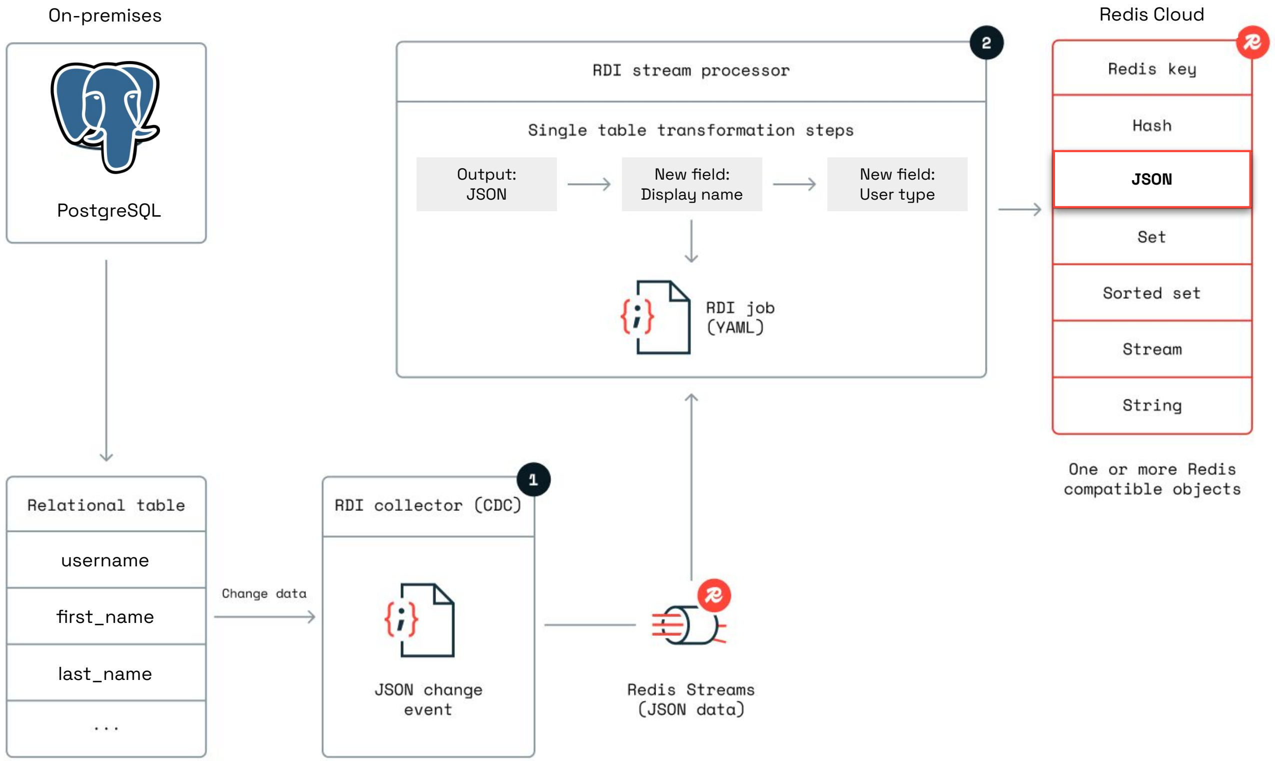
→ Capture, ingest, and transform changed data with configuration, not code

→ Effortlessly deploy, manage, and visualize your data pipeline in Redis Insight or Redis Cloud

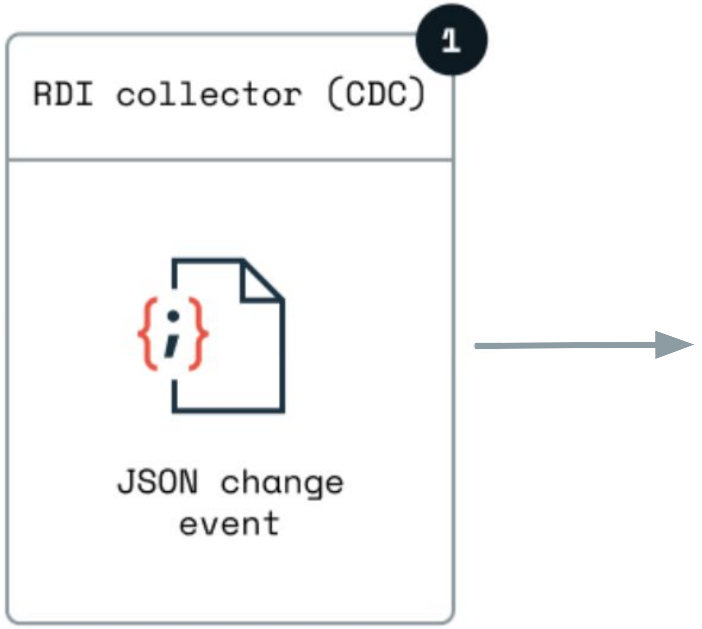

Application
RDI
Stream processor
4 Transformed data stored in target datbase
Redis target database
3 Streams are read and transformations performed
RDI database
Streams
Source database
1 Capture changes or initial snapshot
CDC collector
2 Write to streams in RDI database

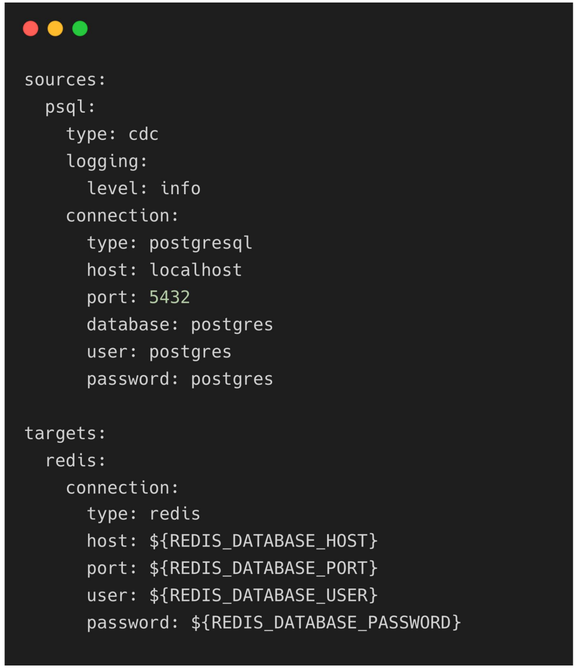
1. **RDI initially populates** Redis with all the data from your database during initial cache loading.
2. **A change data capture (CDC) collector** keeps track of any changed data written to your database.
3. **RDI stream processor** continuously translates the data to the preferred data model and types.
4. **The data is ingested into Redis** and can be read from your app fast.

It’s time for a demo.


On-premises
PostgreSQL
Relational table
username
first_name
last_name
...
Change data
1
RDI collector (CDC)
JSON change event
Redis Streams (JSON data)
2
RDI stream processor
Single table transformation steps
Output: JSON
New field: Display name
New field: User type
RDI job (YAML)
Redis Cloud
Redis key
Hash
JSON
Set
Sorted set
Stream
String
One or more Redis compatible objects

Data pipeline configuration



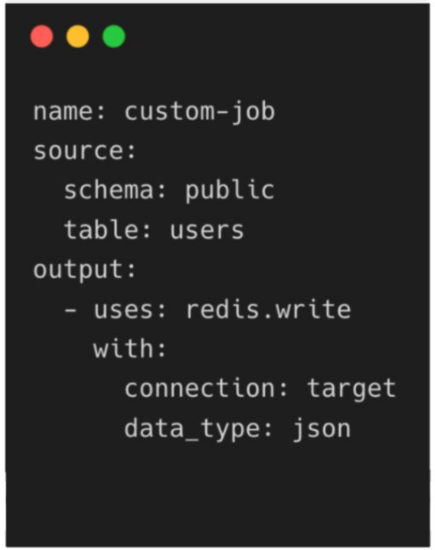


```
sources:
  psql:
    type: cdc
    logging:
      level: info
    connection:
      type: postgresql
      host: localhost
      port: 5432
      database: postgres
      user: postgres
      password: postgres

targets:
  redis:
    connection:
      type: redis
      host: ${REDIS_DATABASE_HOST}
      port: ${REDIS_DATABASE_PORT}
      user: ${REDIS_DATABASE_USER}
      password: ${REDIS_DATABASE_PASSWORD}
```

Managing transformation jobs

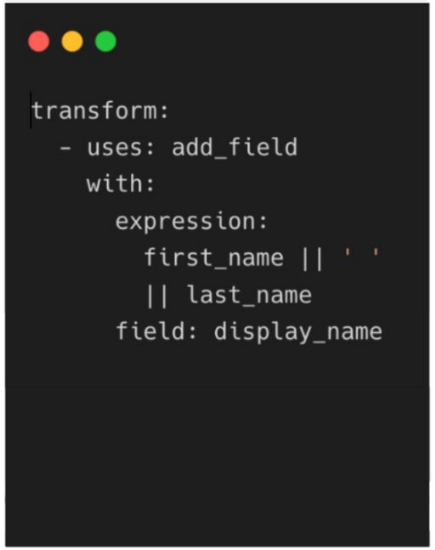
Output:
JSON



```
name: custom-job
source:
  schema: public
  table: users
output:
  - uses: redis.write
    with:
      connection: target
      data_type: json
```

→

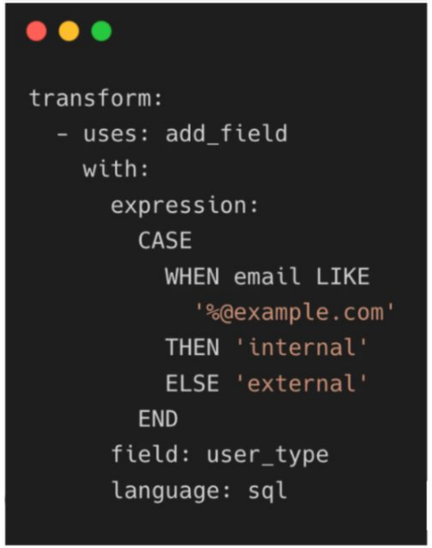
New field:
Display name



```
transform:
  - uses: add_field
    with:
      expression:
        first_name || ' '
        || last_name
      field: display_name
```

→

New field:
User type



```
transform:
  - uses: add_field
    with:
      expression:
        CASE
          WHEN email LIKE
            '%@example.com'
          THEN 'internal'
          ELSE 'external'
        END
      field: user_type
      language: sql
```

Next steps

Today's demo for you to try

Postgres to Redis RDI demo

Try the demo yourself: Clone the repository and follow the setup in the README.

Your best friend: redis.io/dev

Hands-on RDI lab

Two hour hands-on lab: Explore how RDI works, use CDC with PostgreSQL, and deploy pipelines using Redis Insight.

Follow me on LinkedIn

@riferrei

Stay up to date with the latest Redis innovations, join discussions about AI, architecture patterns, and lots of code.

Thank you.