Redis streams in production

Hands-on activity: introduction

Description

For this exercise, we'll look at some example Python code that demonstrates two stream memory management techniques presented in this week's material:

1. Time-based partitioning with expiry, to manage a potentially never ending stream whose rate of growth may change over time.
2. Trimming with **XADD** to maintain a stream at a constant length as new messages are added to it.

This example works for Redis 5 and higher. If you are using Redis 6.2 or higher, you may wish to consider using the **MINID** stream trimming strategy instead of multiple streams. This would keep all of your data in a single stream key and you'd trim the stream periodically to a specific point in time using the **XADD** or **XTRIM** commands. Note however that when you use this approach, you can't spread your data out over members of a Redis cluster as that would require multiple streams in different keys. For more information on the **MINID** stream trimming strategy, see the "Capped Streams" section of the Streams Tutorial on redis.io.

Imagine we have a device that generates a new temperature reading every second. More of these devices could be added to the system in future, in which case we would expect to receive several readings per second. The readings are used to calculate hourly average temperatures, which are displayed on a dashboard and stored in a data warehouse. There is no requirement for long-term storage of the raw temperature readings in Redis Streams.

Stream Implementation Choices

Our example implementation uses two data streams:

- An incoming stream of raw temperature readings. These will arrive at one-second intervals from each producer. There could potentially be many producers. The data is important, but does not need to be kept forever.

- A stream of hourly average temperature values, calculated from data in the raw temperature readings stream. There will always be one value calculated and placed into this stream for every hour of raw temperature data gathered, no matter how many producers are generating readings. We'd like to always have about five days worth of hourly average temperature data available in the stream.

For the raw temperature readings, we'll use a time-partitioned stream with expiry strategy as follows:

- Every day, a new stream will be created and named **temps:YYYYMMDD**. For example a stream for the 1st of January 2025 would be called **temps:20250101**.
- Each time a new entry is written to the stream, the stream's expiry time is extended two days into the future from the time of the message that was written.
- With this approach, our data set is represented as a rolling set of streams where the oldest expire automatically to be replaced by new ones.
- Implementing this strategy will require both the producer and consumer code bases to understand the stream naming strategy.

For the hourly average temperature values, we'll use a single stream whose length is capped as new messages are placed onto the stream. This strategy need only be implemented in the producer code base, as the stream's length can be trimmed using the `MAXLEN` modifier to the `XADD` command.

Code

The code is contained in two Python files, both in the folder **src/week4**.

partitioned_stream_producer.py contains the producer code which will generate a sample temperature data set across a number of stream partitions.

stream_consumers.py contains code for two consumer processes:

- The aggregating consumer: This process reads from the partitioned streams of temperature data and calculates hourly averages. It then acts as a producer, placing the hourly averages on another stream whose length is capped with `XADD`.
- The averages consumer: This process simply reads from the hourly averages stream and displays values from the messages in it.

Before each run, the producer code will reset all streams and other Redis keys that the code references. This example code uses the Python Redis Client to connect to Redis and issue commands.

← Previous

Next →

Modules

Course overview

- Lesson

  Course overview

- Lesson

  Environment setup

Advanced consumer group management

- Lesson

  Managing pending messages

- Assessment

  Quiz 1 | Redis streams in production

- Lesson

  Consumer recovery & poison-pill messages

- Assessment

  Quiz 2 | Redis streams in production

- Lesson

  The XAUTOCLAIM command

Performance and memory management

- Lesson