Redis streams in production

Running the consumers

Module: Running the Consumers

This exercise assumes that you have cloned the course GitHub repository and followed the setup instructions. Note that you will need to activate your Python virtual environment and set any required environment variables for each new terminal session that you use here.

The two consumer processes are both contained in the same file **stream_consumers.py**.

Start them both using a single command in one of your lab environment's terminal sessions:

cd src/week4

python stream_consumers.py temps:20250101

We pass in the parameter **temps:20250101** to tell the aggregating consumer where to start from, as it needs to be seeded with an initial stream partition name.

You should see the output similar to the following:

agg: Starting aggregating consumer in stream temps:20250101 at message 0.

avg: Average temperature for 2025/01/01 at 0 was 23F (3600 observations).

avg: Average temperature for 2025/01/01 at 1 was 51F (3600 observations).

avg: Average temperature for 2025/01/01 at 2 was 84F (3600 observations).

avg: Average temperature for 2025/01/01 at 3 was 73F (3600

observations).

...

avg: Average temperature for 2025/01/01 at 21 was 61F (3600 observations).

avg: Average temperature for 2025/01/01 at 22 was 54F (3600 observations).

avg: Waiting for new messages in stream temps:averages

agg: Changing partition to consume stream: temps:20250102

avg: Average temperature for 2025/01/01 at 23 was 63F (3600 observations).

avg: Average temperature for 2025/01/02 at 0 was 80F (3600 observations).

...

Output from the aggregating consumer appears on lines beginning with **agg**. Output from the averages consumer appears on lines beginning **avg** and is colored yellow in your terminal.

Allow the consumers to run for long enough to process all of the data, which may take some time. Once they have done so, they will wait in a blocking loop, which looks like this:

avg: Waiting for new messages in stream temps:averages

agg: Waiting for new messages in stream temps:20250110, or new stream partition.

You can then stop the consumers with Ctrl + C.

Let's look at what both of these processes are doing in more detail.

The Aggregating Consumer

This consumer reads from the stream partitions that the producer created. Its job is to calculate the average temperature for each hour then place a message containing that information into a second stream called **temps:averages**. It initially needs to know which stream partition containing raw temperature data to begin from, and that is provided via command line arguments when starting the script.

Once it has processed all the messages in its initial stream partition, the consumer blocks and waits for one of two things to happen:

More messages to appear in the stream (meaning there are more readings for the day that it is currently processing).

OR

A new stream partition for the next day becomes available, in which case it knows that it has finished processing the current day's stream and should attach to the one for the next day and process that. The consumer's implementation understands the stream naming convention that the producer uses for partitioning the data, so it knows for example that once all the data in **temps:20250101** has been processed, it should start processing data in **temps:20250102** when that becomes available.

The average temperature for an hour is calculated by looking at the timestamp IDs of the messages in the stream, and totaling up the temperature values in the message payloads until a timestamp representing the next hour is seen. Once the average for an hour has been calculated, it is placed on the **temps:averages** stream. The length of the **temp:averages** stream is also capped at this point using the **MAXLEN** modifier to **XADD**.

The aggregating consumer uses also uses Redis to store its state, in case it crashes or is stopped and needs to resume.

The Averages Consumer

The second consumer is similar to those we have seen in previous hands-on exercises. It simply uses blocking **XREAD** calls to read anything that is placed on the **temps:averages** stream, outputting the contents of the messages to the console. So that it can be more easily distinguished from the aggregating consumer's output, the averages consumer logs appear in yellow.

This consumer also uses Redis to store its state in case of a crash.

← 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