**Title:** Redis XREAD

***Excerpt:***Redis introduced the streams from version 5.0 which mimics the log file data structure in a more advanced manner. The XREAD is used to consume data from Redis streams. It can be used in both non-blocking and blocking manners. Most importantly, the XREAD command can read data from multiple streams at the same time.

**Permalink:** redis-xread

**Category:** Redis

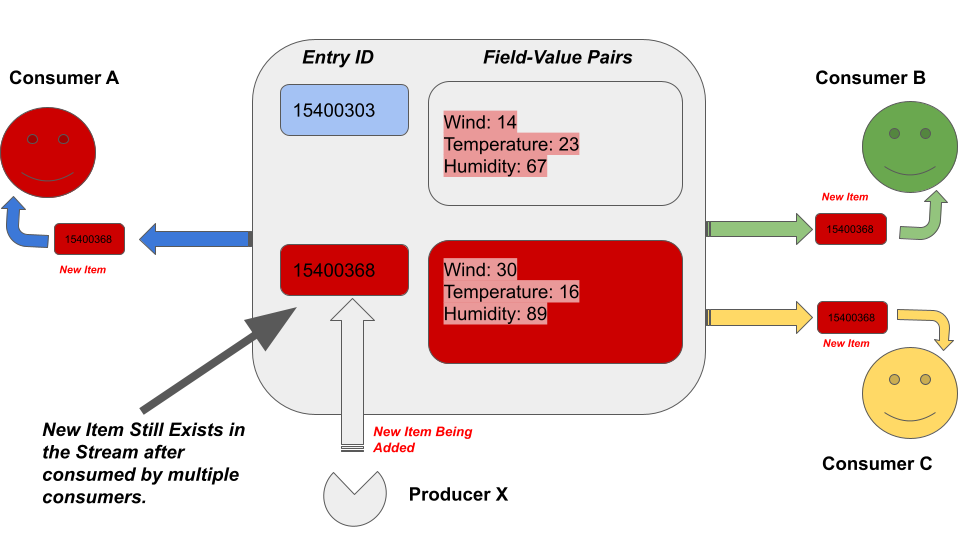
# 

# 

# 

# Redis Streams, Consumers, and Blocking Operations

With version 5.0, Redis introduced streams that mimic the log data structure. Stream is an append-only data structure with a richer set of operations than in a log file. It is one of the most complex data types in Redis since it implements additional blocking operations which allow clients to wait for the new stream data. This is somewhat similar to the behavior of Redis Pub/Sub or blocking lists but fundamental differences are there when it comes to how the consumers consume Redis stream data.



As shown in the above illustration, several advantages over Redis Pub/Sub and blocking lists can be seen. Every new data item will be delivered to every consumer and unlike in the lists removing the list item whenever called to BLPOP or BRPOP, the stream items will remain as it is in the stream. XREAD command operates as a blocking and non-blocking candidate on Redis streams.

# The XREAD Command

The XREAD command can fetch entries from multiple streams simultaneously while the returned entries have an ID bigger than the last received ID for a given consumer. It can operate in both the blocking and non-blocking manner. In the non-blocking nature, the command behaves very similar to the XRANGE command but with some additional features listed in the following.

* It can Fetch entries starting from the most recent entry that has the greatest ID than any other item in the stream.
* It can read from multiple streams at the same time.

This command has a linear time complexity when the N number of elements are stored in the stream. Hence, with a fixed return count, the time complexity would be a constant.

The XREAD command follows the below syntax.

## Syntax:

| XREAD [COUNT number\_of\_returned\_elements] [BLOCK blocking\_time\_in\_milliseconds] STREAMS key [key ...] id [id...] |
| --- |

***COUNT <number\_of\_returned\_elements>*:** The number of elements to be returned by the command. It limits the returned rows to a specified number.

**BLOCK <blocking\_time\_in\_milliseconds>:** The maximum time to wait for a new item to appear in the stream.

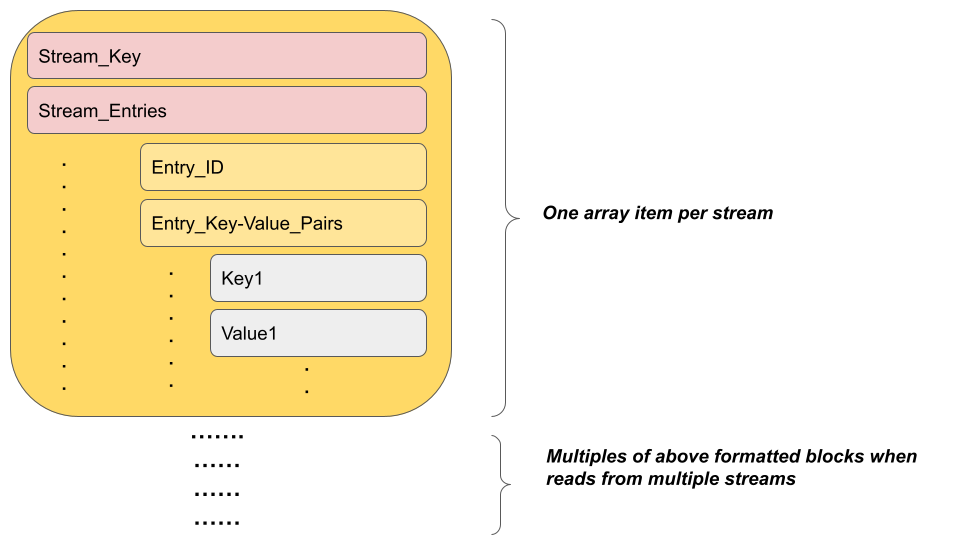
The above two options are optional to the command.

**STREAMS <key>:** The key of the stream. This is a mandatory option and must be the last option in the command since it accepts the variable length of keys and entry IDs.

**<id>:** The ID of the stream entry.

Multiple keys can be specified since the command allows you to read from more than one stream. At the same time, multiple IDs can be provided.

This command returns an array reply. Each array item consists of two elements as shown in the following format.

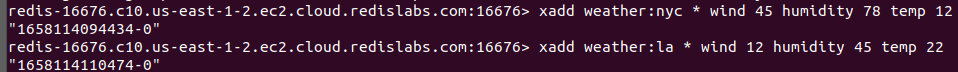


## Example 01 - Inspect Weather Data for Two Locations with Non-Blocking XREAD

Let’s assume that we got two streams containing weather data for LA and NYC. In our weather data publication site, we need to consume from both the streams and fetch the latest weather data for these two locations. The XREAD command is the ideal candidate to use in this scenario with its non-blocking variant.

It's time to create two streams named *weather:nyc* and *weather:la* and populate a couple of entries with some field-value pairs as shown in the following.

| xadd weather:nyc \* wind 45 humidity 78 temp 12  xadd weather:la \* wind 12 humidity 45 temp 22 |
| --- |



Both the streams *weather:nyc* and *weather:lc* have been created successfully and the returned entry IDs are *1658114094434-0* and *1658114110474-0* respectively.

Let’s use the XREAD command to read from both streams at the same time in a non-blocking manner.

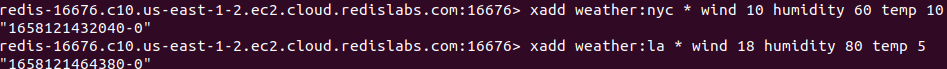
| xread streams weather:nyc weather:la 0 0 |
| --- |

As expected, the output contains the entries from both streams with the ID sequence starting from 0. It is acceptable to specify incomplete IDs as above where both the IDs are 0 which is the millisecond timestamp without the sequence number part. Hence, the above command can be written as in the following.

| xread streams weather:nyc weather:la 0-0 0-0 |
| --- |

Let’s add a couple of entries to both the streams now.

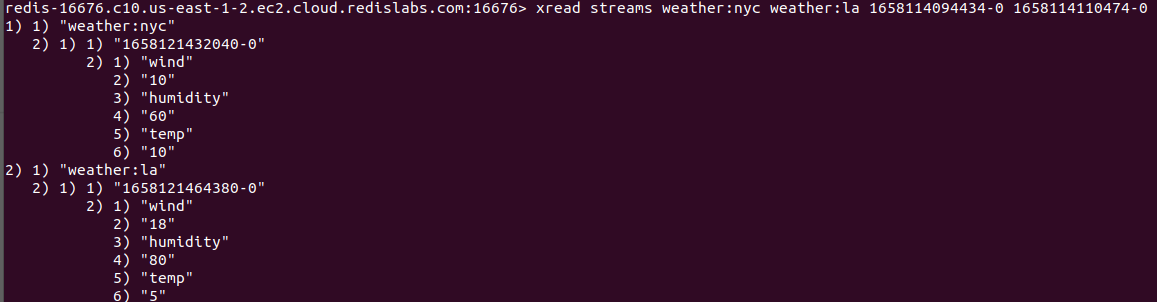
| xadd weather:nyc \* wind 10 humidity 60 temp 10 xadd weather:la \* wind 18 humidity 80 temp 5 |
| --- |



Since we already have the latest entry IDs for both the streams from previous commands, let’s call the XREAD command again to fetch all the entries with bigger IDs than the ones we already queried previously.

| xread streams weather:nyc weather:la 1658114094434-0 1658114110474-0 |
| --- |

As you could see that the specified IDs are from the previous query. Now the command call will return all the entries which have greater IDs than the specified ones.



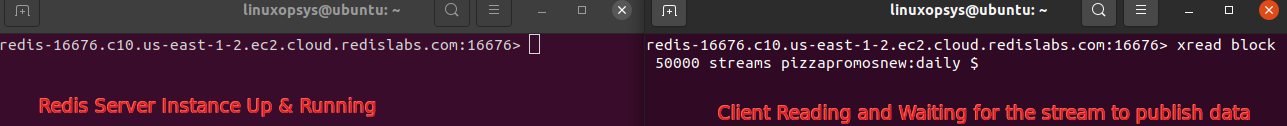
As you can see the newly added entries have been returned from the above command. Next, what you can do is take the entry IDs returned from the above command and call the XREAD with those IDs until the returned array is empty.

## Example 02 - Get the Latest Pizza Promos with Blocking XREAD

There is another variant of the XREAD command which can be used to wait until the publishers publish new data to the stream without terminating immediately as a non-blocking call. Let’s assume a scenario where the pizza guys want to push notifications to all the customers regarding the latest promos available. There might be no promos on certain days. Hence, the customers should wait until new promos are available. It can be achieved with the XREAD command with the block option in place.

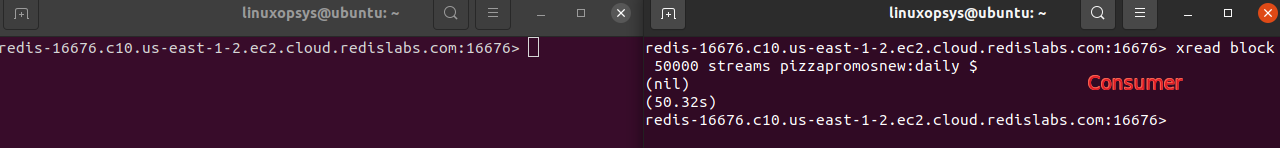
Let’s assume that the pizza company is publishing promo details to a stream called *pizzapromos:daily.* Hence, we can use the XREAD command to wait until a new promo item is added to the stream.

| xread block 50000 streams pizzapromosnew:daily $ |
| --- |

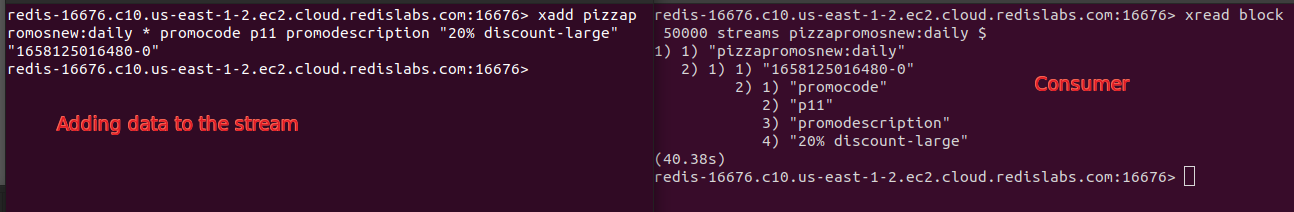


In this case, we have specified the entry ID as $ which is interpreted as the top entry id. Hence, the command will only query only the new entries added to the stream and not the historical entries.

Since we have not added new entries to the stream it will timeout after 50000 milliseconds with a *nil* return as shown in the following.



Now let’s add an entry to the stream using the XADD while another consumer is waiting for the data with the XREAD command as shown in the following.



As expected, the added entry has been consumed by the consumer immediately. From the next call, we need to make sure that we pass the ID that is returned from this command and not the $. If not we will miss the entries added in between.

If multiple clients are waiting for the same stream, the newly added data will be pushed to all of them immediately. The XREAD command is a very useful and recommended command to use in blocking nature applications.

# Conclusion

To summarize, the XREAD command is one of the widely used commands that operate on Redis streams. It can operate in both blocking and non-blocking ways. As discussed, the non-blocking variant is very much similar to the XRANGE command with a couple of differences. In addition, this command can be used with the block option to wait till publishers publish new data to the stream. Overall, the XREAD command is specialized in consuming data from multiple streams simultaneously. It is a nice feature that modern-day applications are looking for.