**Title:** Redis XACK

**Excerpt:** Redis supports stream data type where multiple consumers can read data from such type. The consumer group technique allows multiple consumers in a specific group to read from the same stream and process messages. The XACK command is used to process messages by a consumer in the consumer group. Upon calling the XACK command for a given entry, the PEL entry will be removed for that entry.

**Permalink:** redis-xack

**Category:** Redis

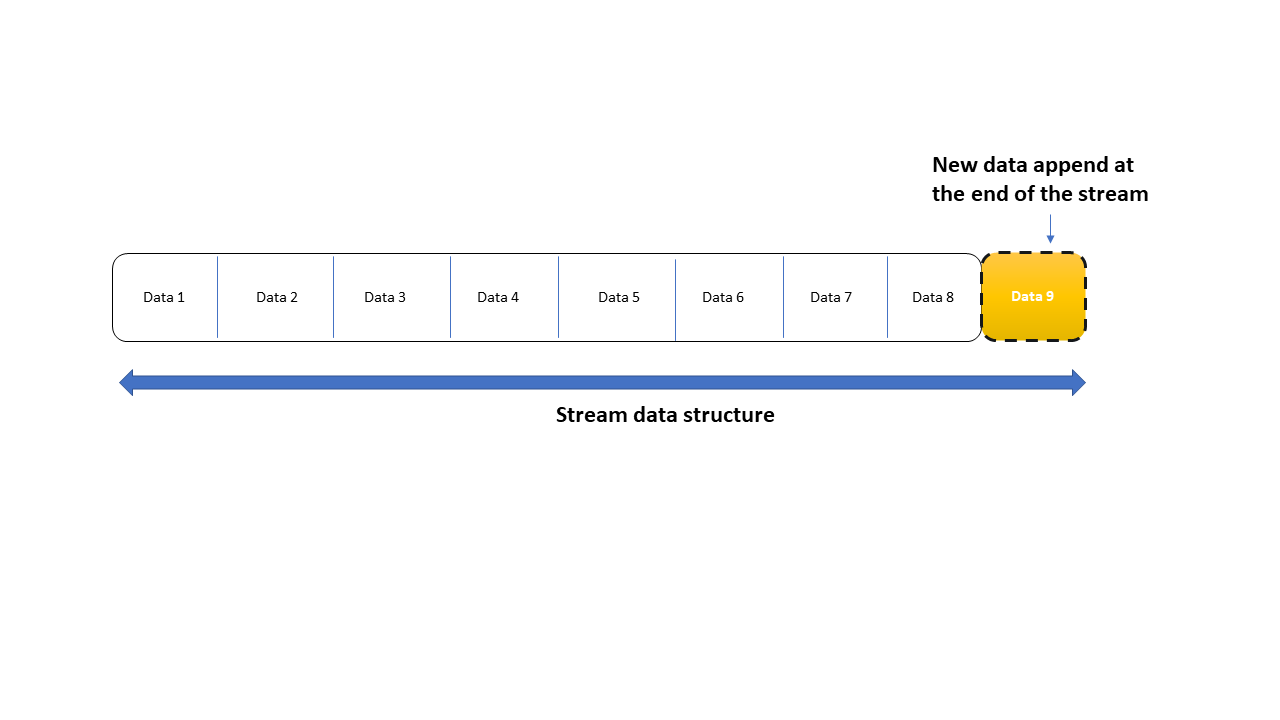
# 

# 

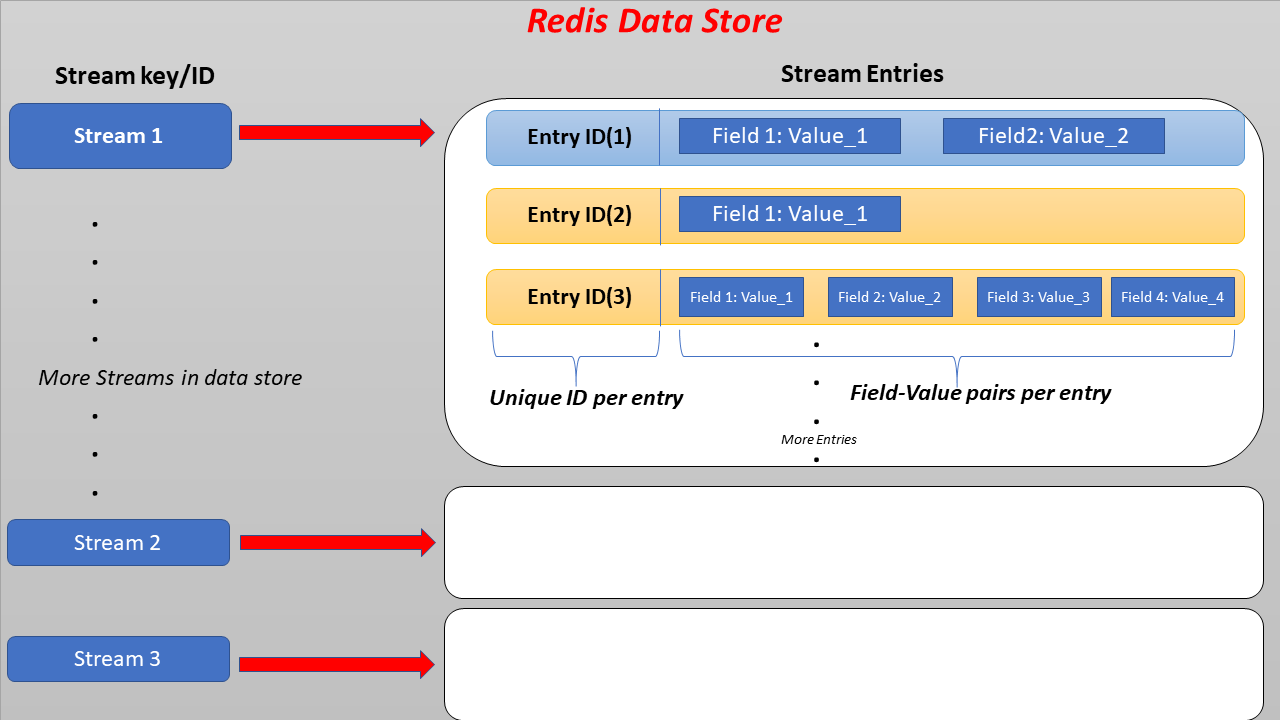
# 

# Introduction to Redis streams

Redis includes common data types like strings, lists, hashmaps, and sets. In addition, it supports more complex data structures like streams. It has been introduced from the Redis version 5.0. The stream type is based on the widely known log data structure. Hence, Redis streams use the append-only technique in their base implementation.



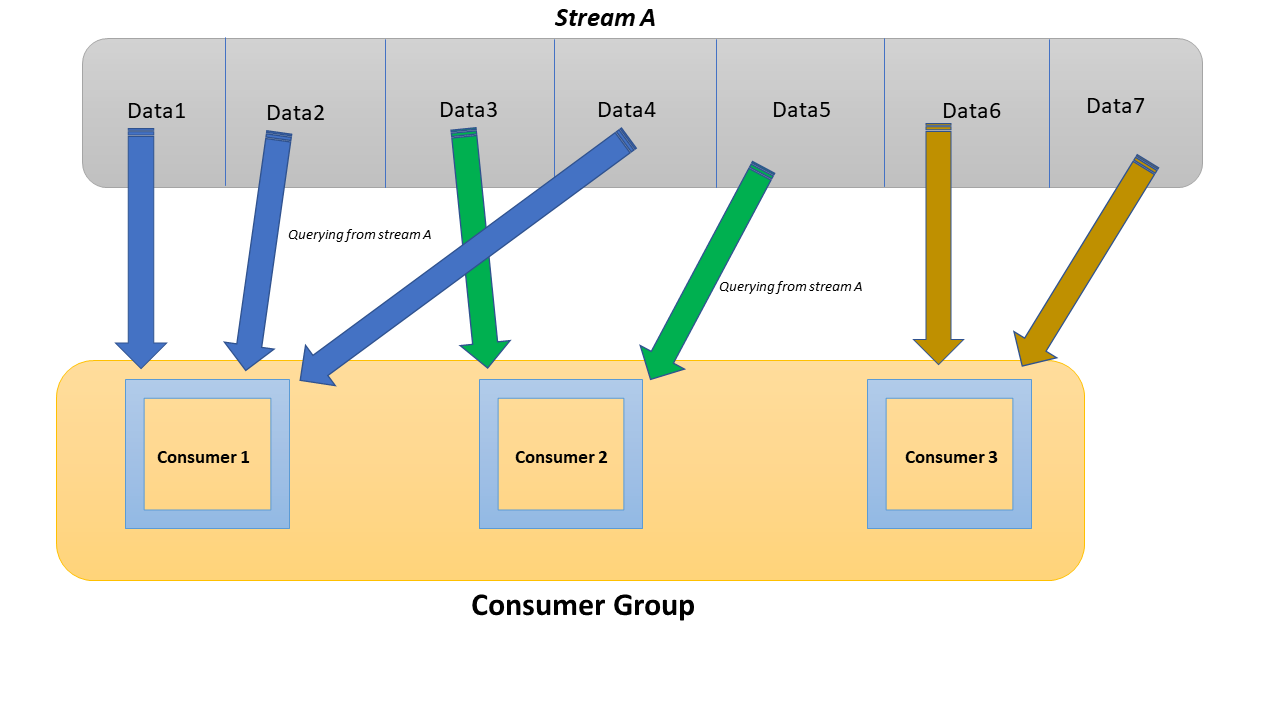
The data in the stream is immutable. The new data can only be appended at the end of the stream. Redis streams can hold entries that are not just strings. Each entry can contain one or multiple field-value pairs like in Redis hashes. Those entries have a unique ID to identify them within the stream that is similar to the line numbers or byte offsets used in a log file. The following illustration will give a good understanding of what the Redis stream looks like.



The XADD command is used to add entries to a given stream which is straightforward. The data access mechanism is different with stream types compared to other types. The primary benefit of streams is that they can push the newly appended messages to multiple clients or consumers. It is only one way of looking at Redis streams. Furthermore, we can see it as a time-series store where you can iterate over the whole stream to fetch all the entries for a given time frame.

# Redis stream consumer groups

As mentioned, Redis streams allow multiple consumers to read data from it. In addition, it extends this functionality to a level of accessing a subset of the stream messages by different consumers. Each consumer will catch different data to process whereas Kafka implements the same behavior with consumer groups. The consumer groups technique is available in Redis streams that allow distributing the given stream data among multiple consumers.



We can use the XREADGROUP command to read data via a consumer group. Each consumer group can contain multiple consumers that are identified by a unique name.

# The XACK command

As stated above, consumers within a consumer group get messages from the stream where the message IDs are greater than the last delivered id. Upon the delivery of a message to a consumer, its status will be set to pending and stored in the pending entries list(PEL) of a consumer group. It is a side effect of calling either the XREADGROUP or XCLAIM commands. This would still cause the server to return the pending messages whenever making a call using the XREADGROUP command to fetch the message history per consumer. Hence, Redis consumer groups introduced a message acknowledgment process. The XACK command can be used to notify the server that a fetched message has been successfully processed. It would remove the entry for such a message in PEL.

## Syntax

| XACK <**stream\_key**> <**consumer\_group**> <**entry\_id**> |
| --- |

The XACK command returns the number of acknowledged entries as a reply.

## Example - Load balancer serves different clients from multiple server nodes

Let’s take a look at a real-world scenario where a load balancer reads from a stream of client IP addresses and serves each client from different server nodes. We can think of this as a consumer group reading from a Redis stream where the group contains multiple consumer nodes.

First, we should create a consumer group to which each of our server nodes belongs. We can use the XGROUP command to create a consumer group for a given stream.

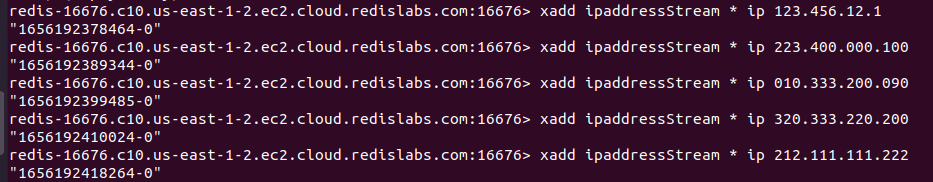
| xgroup create ipaddressStream ukServerGroup $ MKSTREAM |
| --- |

The command is self-explanatory where the *ipaddressStream* has a new consumer group called *ukServerGroup* that provides only the newest messages available to the stream when a consumer is connected. It creates the *ipaddressStream* stream as well, because the *MKSTREAM* parameter has been specified.

Then, we should add some IP addresses to the *ipaddressStream* created previously using the Redis XADD command.

| xadd ipaddressStream \* ip 123.456.12.1 xadd ipaddressStream \* ip 223.400.000.100 xadd ipaddressStream \* ip 010.333.200.090 xadd ipaddressStream \* ip 320.333.220.200 xadd ipaddressStream \* ip 212.111.111.222 |
| --- |

Output after each command:

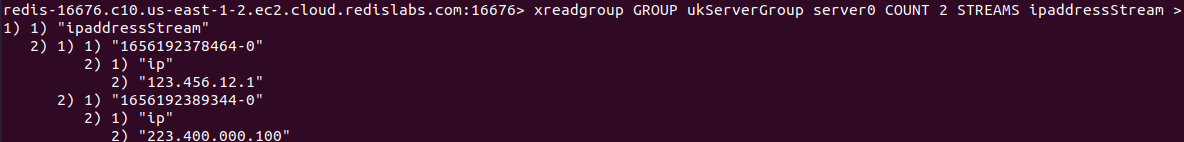


This would add 5 entries to the *ipaddressStream* stream. Each entry is assigned to a server-generated unique ID that has returned after calling the XADD command.

Let’s read the *ipaddressStream* via the *ukServerGroup*’s consumer called *server0*.

| xreadgroup GROUP ukServerGroup server0 COUNT 2 STREAMS ipaddressStream > |
| --- |

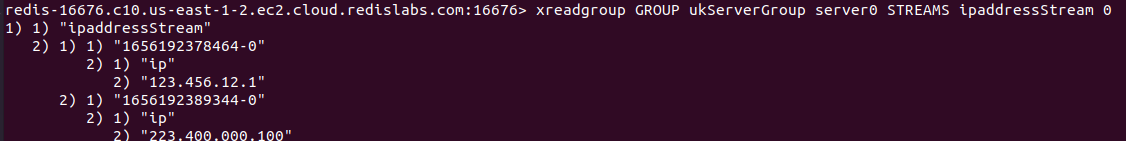
Output:



As expected the *server0* consumer gets two new messages from the *ipaddressStream* stream. These two ip addresses have been added to the pending entries list. If we call the XREADGROUP command with the 0 ID, it will return the pending messages for *server0* consumer.

| xreadgroup GROUP ukServerGroup server0 STREAMS ipaddressStream 0 |
| --- |

Output:



This means the server is still waiting for the *server0* consumer to acknowledge these two messages. Let’s acknowledge the messages for the *server0* consumer using the XACK command.

| xack ipaddressStream ukServerGroup 1656192378464-0 1656192389344-0 |
| --- |

Here we are acknowledging both the entries identified by the respective IDs. The command returns the count of the successfully processed messages as well. It is two in this case.

Output:



After the above process, those two messages should have been removed from the pending entries list(PEL). Hence, the *server0* consumer will not return any pending messages upon calling the XREADGROUP command via the consumer group *ukServerGroup.*

| xreadgroup GROUP ukServerGroup server0 STREAMS ipaddressStream 0 |
| --- |

Output:



It returns an empty array which means no pending messages for this consumer. The message acknowledgment feature is very useful in those types of scenarios.

# Conclusion

Redis comes with the stream data type whose underlying implementation is based on the log data structure. Hence, the new entries are appended at the end of the stream. The biggest advantage is that multiple consumers can query the latest messages added to the stream. Furthermore, the Redis consumer group technique allows the reading stream by a group of consumers where each consumer sees only a subset of the stream messages. Upon reading an entry from the stream by a consumer, such entry is added to the pending entries list. Hence, the consumer needs to acknowledge each of the pending messages. It will notify the server to remove the entry from PEL and release the memory. The XACK command can be used to acknowledge Redis stream messages. It supports acknowledging multiple messages at once.