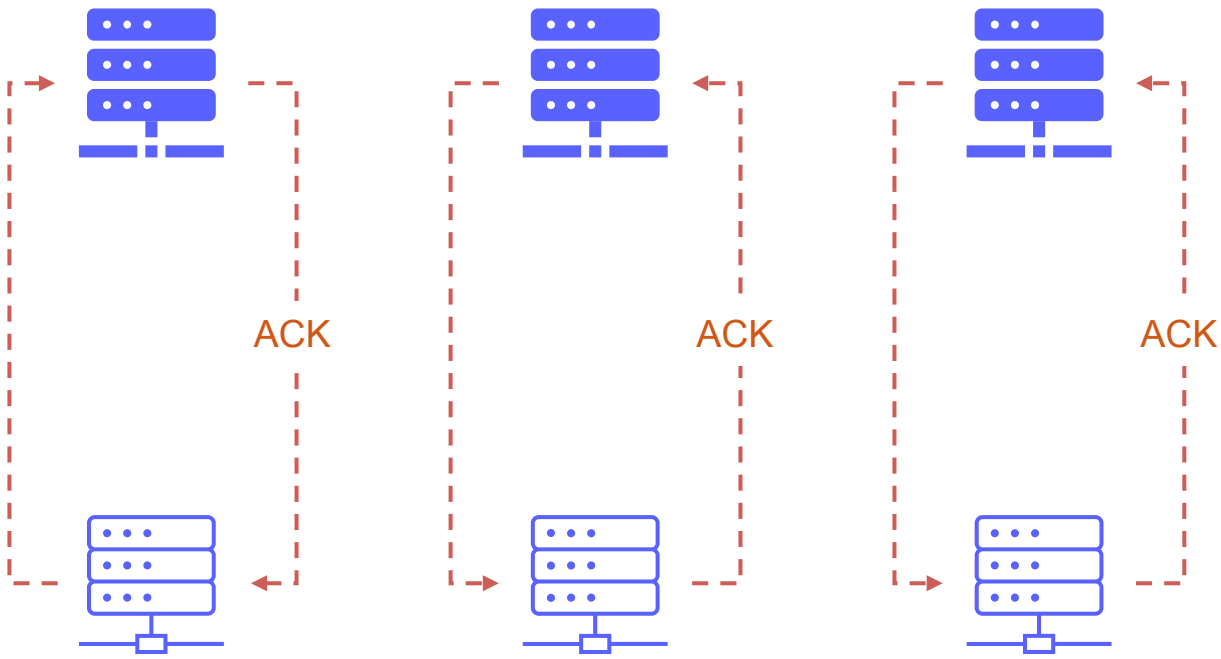
100 XP

Streams

7 minutes

Distributed microservices may often wish to leave data or messages for other microservices to consume. Many different consumer microservices may wish to consume the same messages in a large enough solution. This pattern is commonly referred to as competing consumers architecture.

In the retail company scenario, microservices must leave messages asynchronously for many clients to consume. Ideally, these clients would consume this data at their schedule without forcing the message producer to wait. Today, these microservices are designed to send data and wait for a reply directly, or acknowledgment, to continue processing. As the retail solution scales, it becomes increasingly important to design and implement a solution where message producers and consumers can asynchronously store and parse messages without waiting for external stimuli.

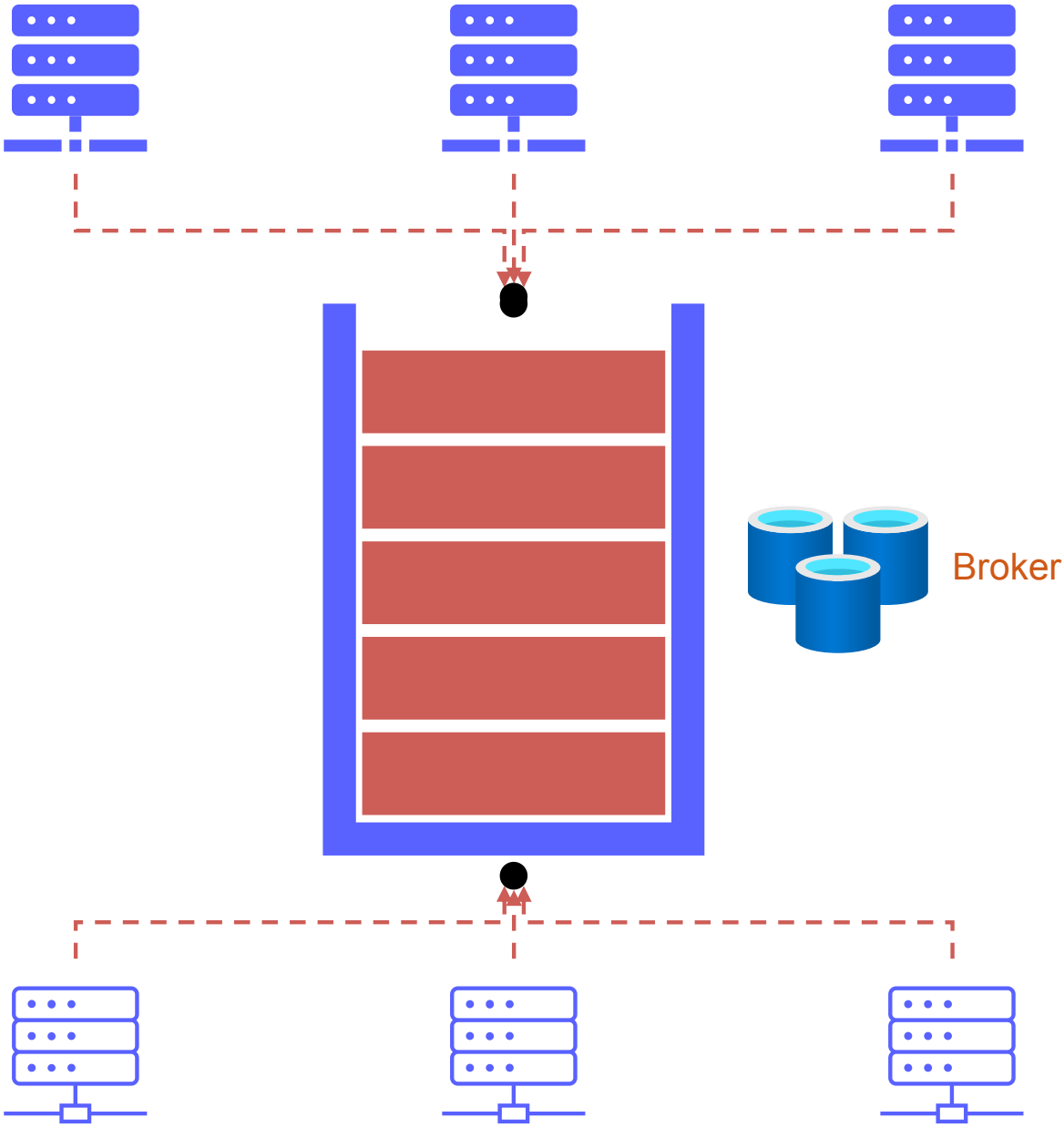


Message brokering using Streams

A *message* broker is a middleware solution that manages the distribution of message data to multiple consumers that would typically compete for the same data.

Here, you learn how the Streams feature of Azure Cache for Redis can serve as middleware to simplify the storage of messages from multiple producers and the distribution of messages to many clients. Streams can help your application components manage messaging in a scalable, fault-tolerant, and distributed manner.

The Streams feature of Azure Cache for Redis stores messages in a stream that clients can consume. Microservices can use this feature to distribute messages to many clients that the microservice may not know about at the time of message production. Additionally, the number of message consumers and producers can be scaled up or down separately in patterns that make sense for a cloud solution.



This feature makes the brokering of messages between microservices simpler.

Adding entries to a stream

Clients can produce messages for a stream using a set of field-value pairs. The client can specify a unique identifier when the message is created or have it autogenerated by Redis. In Redis terminology, the message is referred to as an **entry,** and the unique identifier is the **key**.

> **Note**
>
> An auto-generated key is composed of the Unix time of the local machine in milliseconds and a sequence number if multiple entries occur within the exact millisecond. For example, if the current date and time on the local device is midnight (UTC) on December 1, 2010, and the entry is the third that occurred in the current millisecond, the auto-generated key would be **1291161600000-3**.

Redis includes a `XADD` command used to add new entries to a stream. The stream doesn't need to be created before issuing the command. For example, use the `XADD` command to add a new entry to the **media.photos.genthumb** stream using an autogenerated key and the following field-value pair data:

Expand table

Field	Value
location	*sd7f9sd7.png*
width	*300*
height	*300*

Redis

```
XADD media.photos.genthumb * location sd7f9sd7.png width 300 height 300
```

Alternatively, you can specify a specific key to use with the `XADD` command. In this example, the `XADD` command is used to add a new entry to the **media.photos.genthumb** stream with a key of **1596514316945-2** and the following field-value pair data:

Expand table

Field	Value
location	*xczv897.png*
filter	*grayscale*

Redis

```
XADD media.photos.genthumb 1596514316945-2 location xczv897.png filter
grayscale
```

Querying entries in a stream

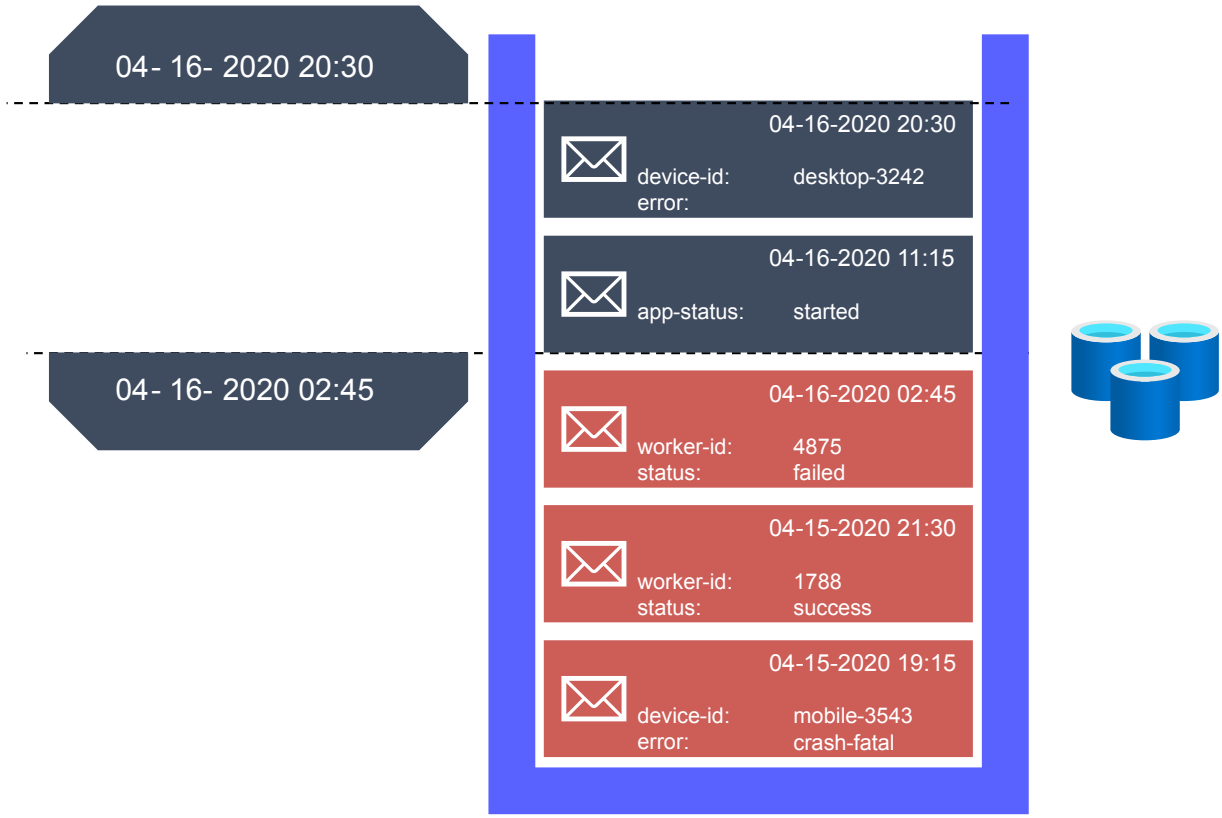
Once there are entries in a stream, clients can manually count the number of entries or retrieve entries directly.

Redis includes an `XRANGE` command to directly get all or a subset of the entries in a stream. Redis also contains an `XLEN` command to count the number of entries in a specific stream.

Querying ranges of entries

The `XRANGE` command is used with three core parameters:

- The name of the stream to query
- The starting point of the result set
- The endpoint point of the result set



The `XRANGE` command supports the use of special operators if you don't know or don't wish to specify the start or end of a range. The `-` operator is used to indicate the range should start from the beginning, chronologically, of the stream. The `+` operator is used to indicate that the range should continue to the end, again chronologically, of the stream.

For a first example, the `XRANGE` command could be used with two keys to retrieve all chronologically occurring entries between the streams. In this code example, the `XRANGE` command is used to get all entries in the **media.photos.genthumb** stream that occurs starting with the **1596514316945-2** key and ending with the **1609476184275-0** key.

Redis

```
XRANGE media.photos.genthumb 1596514316945-2 1609476184275-0
```

As a second example, the `+` and `-` operators can be used together to retrieve all entries from a stream. This example queries the same stream, but retrieves all entries by using the special operators.

Redis

```
XRANGE media.photos.genthumb - +
```

Next, the special operators can be used separately. In this example, use the `-` operator to get all entries from a stream up to the **1596514316945-2** entry.

Redis

```
XRANGE media.photos.genthumb - 1596514316945-2
```

This example illustrates getting all entries starting with the **1596514316945-2** entry to the end of the stream with the `+` operator.

Redis

```
XRANGE media.photos.genthumb 1596514316945-2 +
```

Alternatively, the `XREVRANGE` command is available with the same syntax as the `XRANGE` command to get all entries in reverse order.

Redis

```
XREVRANGE media.photos.genthumb - +
XREVRANGE media.photos.genthumb - 1596514316945-2
XREVRANGE media.photos.genthumb 1596514316945-2 +
```

Counting entries in a stream

The `XLEN` command is a simple command that takes the name of a stream as an argument, and returns an integer with the count of entries within the stream.

Redis

```
XLEN media.photos.genthumb
```

Reading data from a stream

While you can query data from Redis as a time series, it's preferable to listen for new items from a stream. A stream can have many listeners waiting for data stored in the stream. If a new listener is created after the data has started, the listener can still "start over" and process data from the beginning of the stream.

Data is consumed from a stream using the `XREAD` command. This command includes a suite of options that aren't all covered in this module. For now, focus on the `XREAD` command's ability

to read one or more streams, starting from specific points.

Reading data from a specific starting point in a stream

The `XREAD` command, in its simplest and most commonly used form, can subscribe to entries from a stream starting from the first entry. The consumer then processes the entries, in order, before reaching the current end of the stream. At this point, the consumer can wait for new data in the stream.

To subscribe to data in a stream, the `XREAD` command should be used with the `STREAMS` argument and a list of streams to read along with their corresponding starting points.

In this example, the `XREAD` command is used to read data from the **media.photos.genthumb** stream starting from the very beginning by using the **0** key. In most situations, the **0** key is effectively less than the current time-based key.

Redis

```
XREAD STREAMS media.photos.genthumb 0
```

If a stream consumer has to pause consuming messages for any reason, a key can be passed into the `XREAD` command to start consuming messages from a specific point. In this example, messages are consumed, starting with the **1639714145947-2** key.

Redis

```
XREAD STREAMS media.photos.genthumb 1639714145947-2
```

> **Tip**
>
> This technique is convenient if a stream consumer crashes or fails for any reason. The XREAD command can resume reading from the last known good read.

Another option is to use the `$` operator. This particular operator indicates to start reading the stream from the end. Effectively, this example only receives new entries instead of historical entries.

Redis

```
XREAD STREAMS media.photos.genthumb $
```

Reading data from multiple streams

All of the previous examples read data from a single stream. You may remember that the STREAMS argument supports a list of streams along with their corresponding starting points.

In this example, the XREAD command reads from both the **media.photos.genthumb** and the **media.photos.delete** streams simultaneously while starting both streams from their chronologically first entries.

Redis

```
XREAD STREAMS media.photos.genthumb media.photos.delete 0 0
```

Deleting a stream and its data

When data is consumed from a stream, it remains in the stream until it's manually deleted. This implementation allows multiple clients to consume data from the same stream without missing any data.

To manually delete a specific entry from a stream, use the `XDEL` command with the stream's name and the entry's key to delete. In this example, the `XDEL` command is used to delete the entry with a key of **1596514316945-2** from the **media.photos.genthumb** stream.

Redis

```
XDEL media.photos.genthumb 1596514316945-2
```

Removing all items from a stream doesn't delete the stream as an empty stream is an entirely legal construct in Redis.

To delete a stream altogether, use the `DEL` command with the stream's name.

Redis

```
DEL media.photos.genthumb
```

Next unit: Exercise - Broker messages using Streams

Continue >