Streams

RabbitMQ Queues

- RabbitMQ Queues are limited in the following scenarios
  - They deliver the same message to multiple consumers by binding a dedicated queue for each consumer. Clearly, this could create a scalability problem.
  - They erase read messages making it impossible to re-read(replay) them or grab a specific message in the queue.
  - They perform poorly when dealing with millions of messages because they are optimized to gravitate toward an empty state.

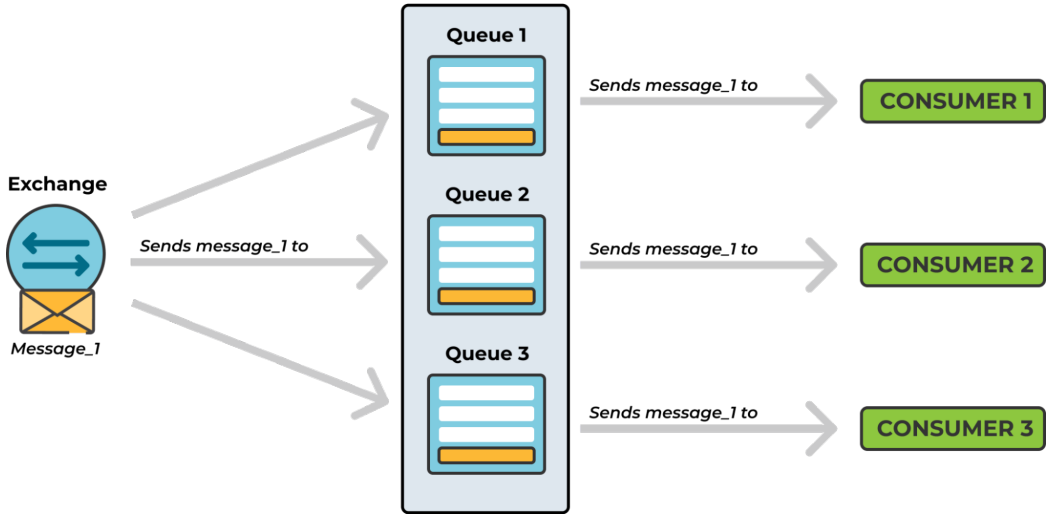
Streams

- Introduced Streams in RabbitMQ 3.9 to mitigate the earlier-listed challenges.
- Streams model an append-only log that's immutable.
- To read messages from a Stream in RabbitMQ, one or more consumers subscribe to it and read the same message as many times as they want.
- Streams are always persistent and replicated.
- Queue type cannot be changed. It must be specified at the time of declaration.

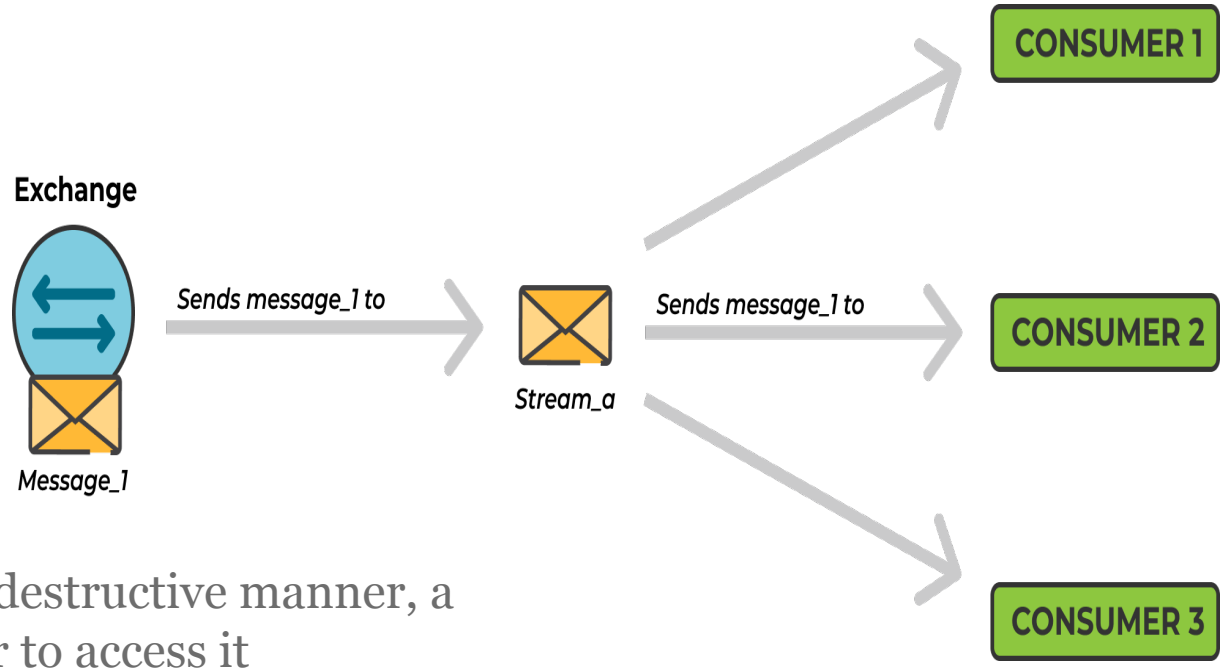
Use Cases RabbitMQ Streams

- **Fan-out architectures:** Where many consumers need to read the same message
- **Replay & time-travel:** Where consumers need to read and reread a message from any point in the stream.
- **Large Volumes of Messages:** Streams are great for use cases where large volumes of messages need to be persisted.
- **High Throughput:** RabbitMQ Streams process relatively higher volumes of messages per second.

Fan-out Architectures



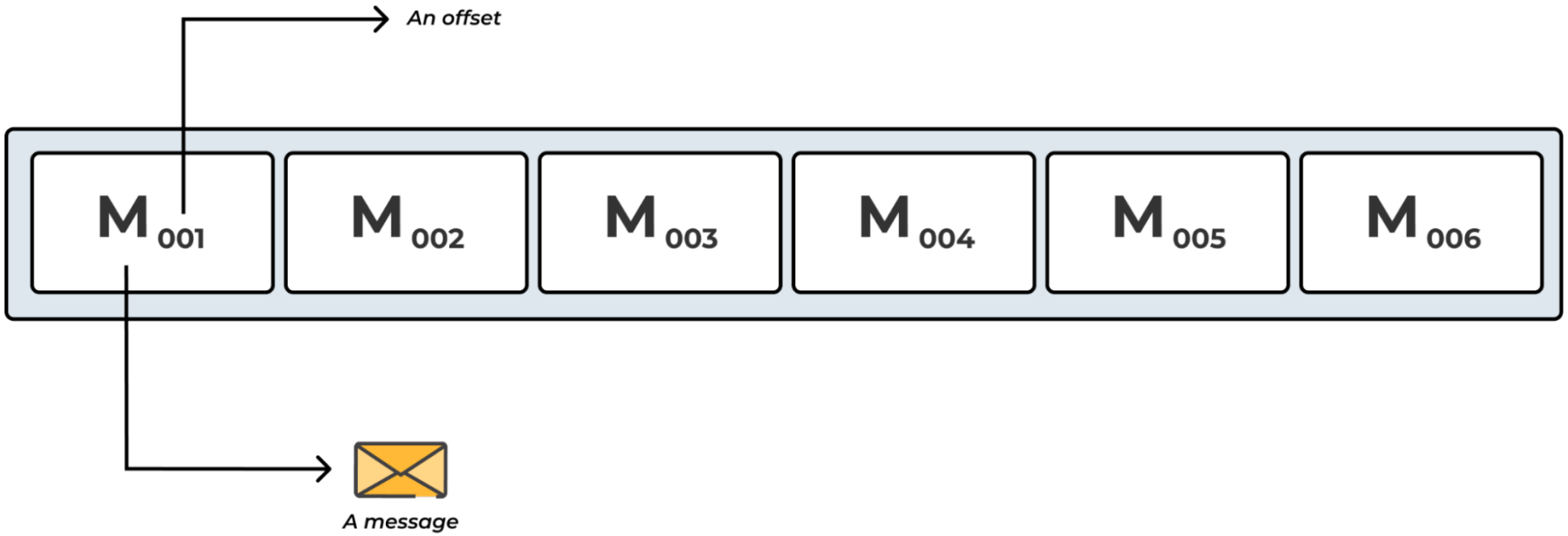
Having to add queues for every added consumer is resource intensive, which gets worse when dealing with queues that need to persist data.



consumers read messages from a Stream in a non-destructive manner, a message will always be there for the next consumer to access it

Replay & Time Travel

- Aside from re-reading the most recent message, it is also possible to re-read a message from any point in the Stream



Large Volumes of Messages

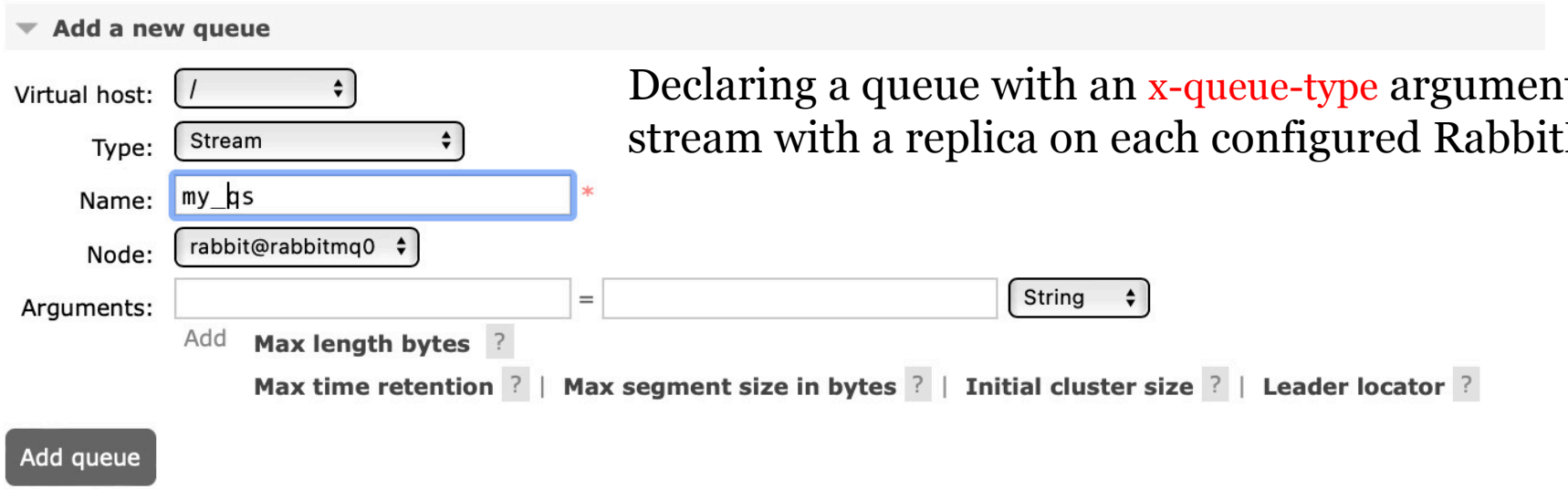
- RabbitMQ Streams are perfect when persisting large volumes of messages. Streams shine in this area because they store messages on the file system.
- As a result, a Stream in RabbitMQ could grow indefinitely until the host disk space runs out.
- RabbitMQ Streams allow setting a maximum log data size. The oldest messages are discarded preventing the Stream from consuming the entire disk space.

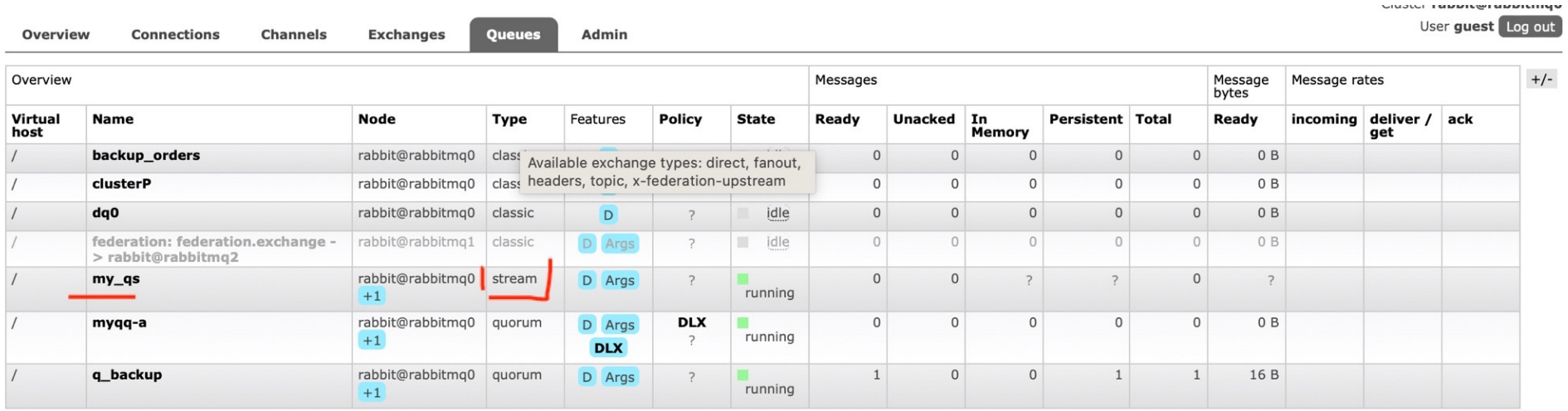
High Throughput

- If a RabbitMQ use case requires processing high volumes of messages per second, then using a Stream is the best option.
- Usually Quorum queues handle about 40,000 messages per second. Streams, on the other hand, handled around 64,000 messages per second when used with AMQP protocol, and over 1 million messages per second when used with native Stream protocol.

Stream

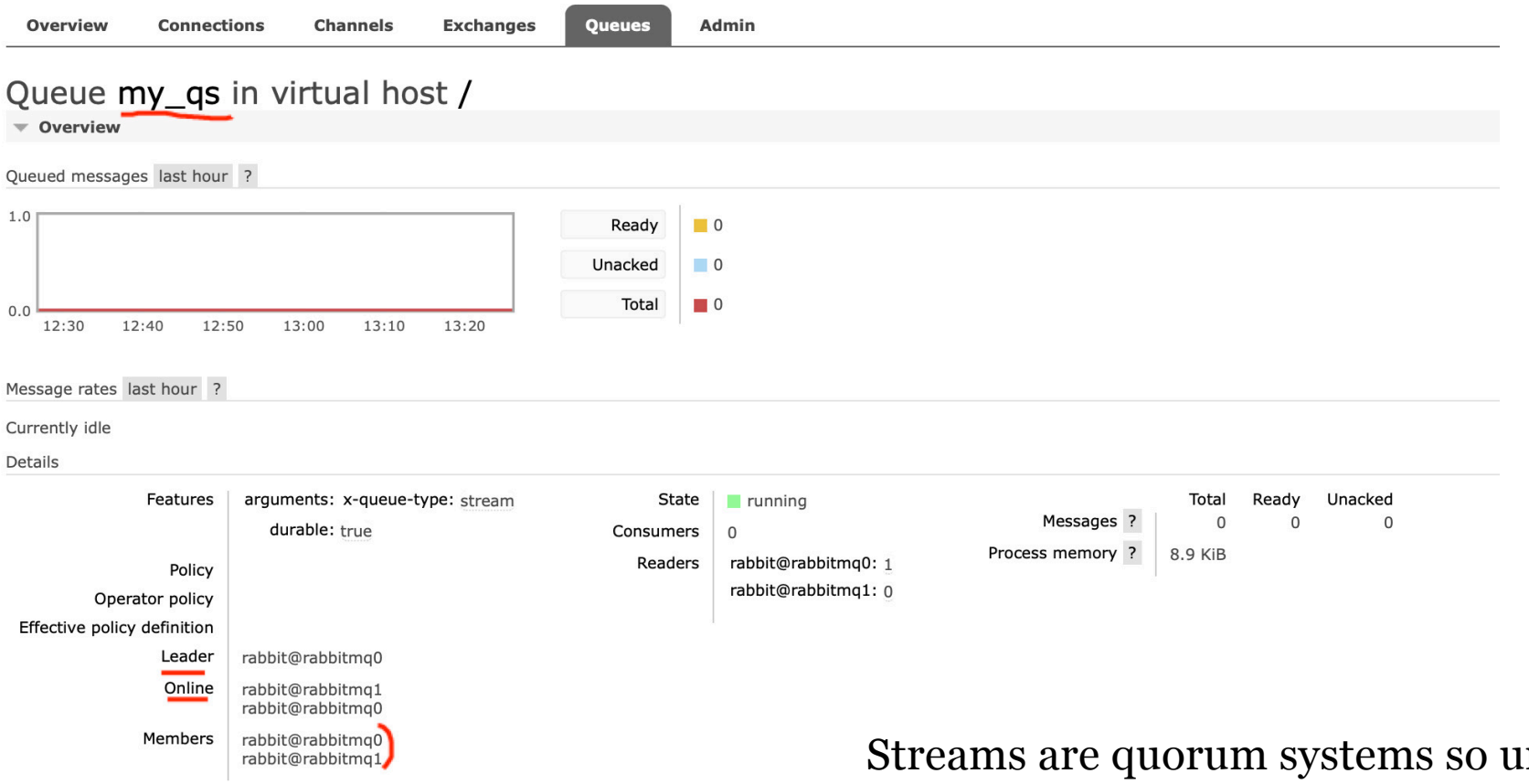
Declaring a queue with an x-queue-type argument set to stream will create a stream with a replica on each configured RabbitMQ node.





Overview							Messages					Message bytes	Message rates			+/-
Virtual host	Name	Node	Type	Features	Policy	State	Ready	Unacked	In Memory	Persistent	Total	Ready	incoming	deliver / get	ack	
/	backup_orders	rabbit@rabbitmq0	class				0	0	0	0	0	0 B				
/	clusterP	rabbit@rabbitmq0	class				0	0	0	0	0	0 B				
/	dq0	rabbit@rabbitmq0	classic	D	?	idle	0	0	0	0	0	0 B				
/	federation: federation.exchange -> rabbit@rabbitmq2	rabbit@rabbitmq1	classic	D Args	?	idle	0	0	0	0	0	0 B				
/	my_qs	rabbit@rabbitmq0 +1	stream	D Args	?	running	0	0	?	?	0	?				
/	myqq-a	rabbit@rabbitmq0 +1	quorum	D Args DLX	DLX ?	running	0	0	0	0	0	0 B				
/	q_backup	rabbit@rabbitmq0 +1	quorum	D Args	?	running	1	0	0	1	1	16 B				

Streams



Streams are quorum systems so uneven cluster sizes is strongly recommended.

Stream - Consumer

```
public static void consume() throws IOException, TimeoutException {
    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost(QUEUE_HOST);
    factory.setPort(PORT);
    factory.setUsername("guest");
    factory.setPassword("guest");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();
    channel.basicQos(100); // QoS must be specified
    channel.basicConsume(
            "my_qs",
            false,
            Collections.singletonMap("x-stream-offset", "first"), // "first" offset specification
            (consumerTag, message) -> {
                System.out.println(" Received : " + new String(message.getBody()));
                channel.basicAck(message.getEnvelope().getDeliveryTag(), false); // ack is required
            },
            consumerTag -> { });
}
```

Streams Configurations

- Stream is broken down into smaller files known as segment files. A Stream truncates its size by deleting a segment file and all its messages. (x-stream-max-segment-size-bytes)
- Stream's retention strategy
  - Size-based retention strategy : x-max-length-bytes
    - the Stream is configured to truncate its size once the total size of the stream reaches a given value.
  - Time-based retention strategy : x-max-age
    - the Stream is configured to truncate a segment file once that segment reaches a given age.

Replication Factor

- Streams are persistent and replicated.
- When a Stream is initialized, RabbitMQ will create a replica of the Stream on some randomly selected nodes in the cluster.

Streams replicas can be controlled in the following ways:

- x-initial-cluster-size queue argument when declaring the Stream via an AMQP client.
  - For example, `"x-initial-cluster-size": 3`
- With the initial-cluster-size queue argument when declaring the Stream via the stream plugin.

Stream Leader Election

- Stream operations go through the leader replica first and then replicated on the other nodes.
  - x-queue-leader-locator argument when declaring the Stream
  - queue-leader-locator policy key
  - queue_leader_locator in the configuration file
- Supported flags:
  - client-local - This is the default value. The client-local value elects this node to be the leader when connected to while creation of Stream.
  - Balanced - Make a balance random node the leader.

Lab : Streams – 60 Minutes