Queues (SQS)

Dr Peadar Grant

October 8, 2021

1 Message queue fundamentals

Messaging queues are a very useful architectural component in many software systems, Figure 1. They can be implemented in many different ways, but the basic ideas remain the same.

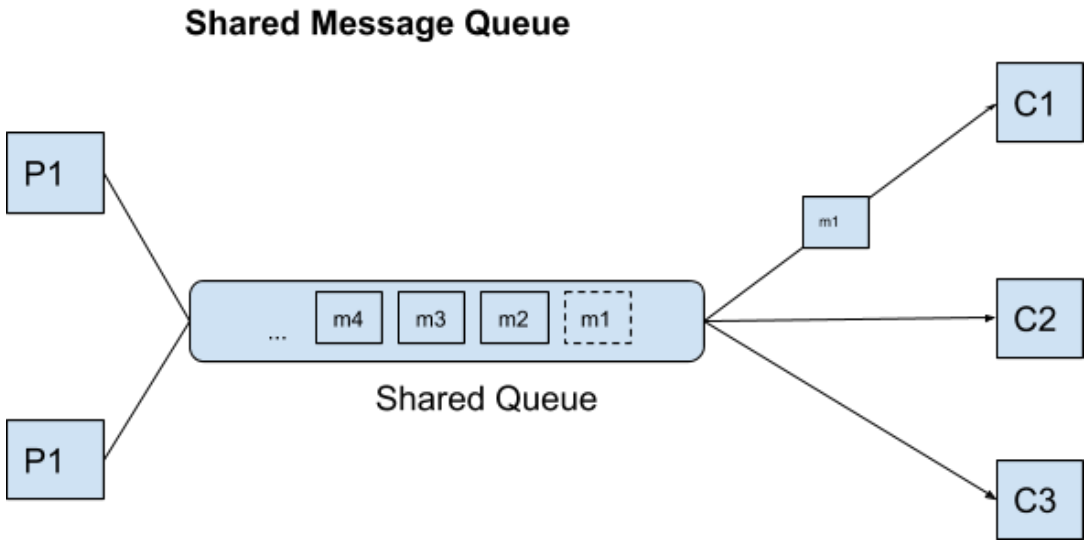

Figure 1: Message queue components

1.1 Components

Message: a piece of data. Could be text (plain, CSV, XML, JSON) or binary (JPEG, audio etc). May have also have attached meta-data. Messages are opaque from SQS.

Queue: a container for related messages and applications

Producer: connects to and places messages into the queue. Producers need not be identical to each other.

Consumer: connects to to and removes messages from the queue. Normally assume that consumers are identical to each other.

Broker: server software that creates queues and listens for connection from producers and consumers.

Note that there can be none, one or many producers and similarly none, one or many consumers attached to the queue at any one time.

1.2 Characteristics

Pull model: consumers connect to the queue and pull messages from it. (This differs from notification systems like SNS topics where messages are pushed to the subscribers.)

Ordering: how does the queue affect the order that messages are delivered in:

First-in first-out (FIFO): where messages progress through the queue in the order they entered it in.

Priority: where messages are ranked and highest-priority messages are delivered first.

Non-specific: where messages are delivered generally in a FIFO manner but may not always be.

Durability: said to be durable if the messages in the queue survive restarting the broker. software.

Time To Live (TTL): if a message remains in the queue longer than the specified time-to-live, then it is discarded. (May be set at a queue level and/or at a message level.)

Re-try / visibility timeouts: what happens if a consumer takes a message and then fails when processing it. Some queues offer features to help re-deliver it.

1.3 Use cases

- **Triggering actions** when something else occurs. Remember that a message in a queue needn't contain all the data about something - often it's just enough to point to data held elsewhere (such as a DBMS).
- **Producer and consumer operate asynchronously** of each other. No need for producer to wait for consumer to process message. (Fundamental difference to a service architecture / API)
- **Consumer may be offline** for short or long periods. Consumer will catch-up when online again.
  - Very useful tool for systems where a central (cloud / data-centre-based) component communicates with a remote component over a poor / intermittent connection.
- **Decoupling of producer and consumer**: both may be written in different languages, run on different operating systems etc. Only requirements is that they both implement the required protocol so can connect to queue via broker.

2 SQS

AWS provides a number of queue options: one of them is Simple Queue Service (SQS), the key features of which are shown in Figure 2.

SQS queues are created thus:

```
# create queue (using defaults) named labq
aws create-queue --queue-name labq

# create queue and capture queue URL in PowerShell
$QueueUrl=(aws create-queue --queue-name labq | ConvertFrom-Json).QueueUrl
# note the format of the Queue URL (it includes your account ID)
```

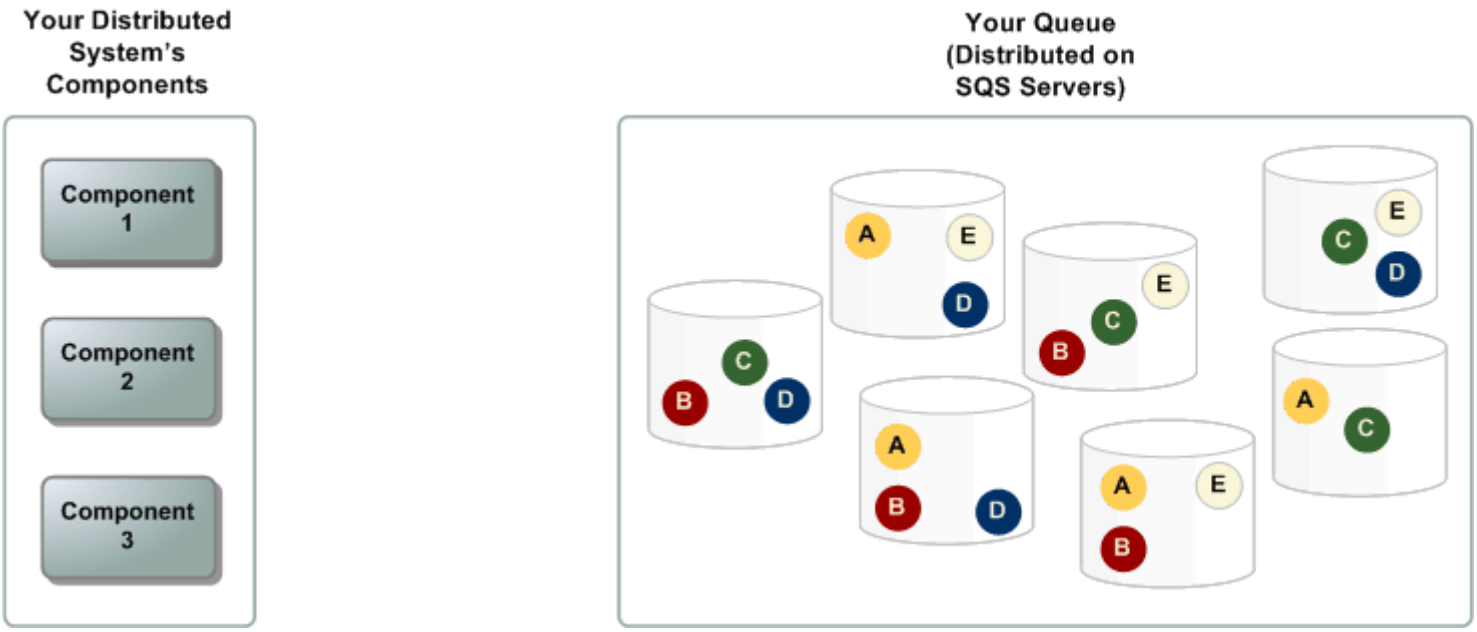

Figure 2: SQS architecture

2.1 Key SQS characteristics

- Protocol is HTTP-based and integrated into AWS CLI and SDK. Easily integrated into different programs.
- Producers and consumers can be programs on your own devices / servers and/or AWS components like Lambda.
- Queue contents are distributed across multiple servers by AWS to ensure redundancy.
- Offered as Platform-as-a-service, so no need to consider broker and underlying components.
- Queues can either be FIFO or "standard queues" where ordering may vary.
- A given message may sometimes be received by more than one consumer. May need to plan logic.

2.2 Sending a message

```
# show command help (to learn options)
aws sqs send-message help

# send a message to a queue given its Queue URL is in $QueueUrl
aws sqs send-message --queue-url $QueueUrl --message-body "HELLO."
# response shows message ID and MD5 hash of message body

# parse the return into PowerShell variables
$SendResult=(aws sqs send-message `
--queue-url $QueueUrl `
--message-body "HELLO." `
| ConvertFrom-Json)

# send a message from file hello.txt to a queue
aws sqs send-message --queue-url $QueueUrl --message-body file://hello.txt
```

2.3 Receiving a message

```
# show command help (first stop always!)
aws sqs receive-message help

# receive message from $QueueUrl
aws sqs receive-message --queue-url $QueueUrl

# capturing as PS vars
$Messages=(aws sqs receive-message --queue-url $QueueUrl | ConvertFrom-Json).Messages
```

Can allow multiple messages to be received in a batch (up to 10):

```
# receive up to 5 messages in one operation
$Messages=(aws sqs receive-message `
--queue-url $QueueUrl `
--max-number-of-messages 5
| ConvertFrom-Json).Messages

# number of messages received
```

2.4 Visibility Timeout

Consumers receive messages off the queue and process them. A consumer might receive a message but fail to process it correctly. Therefore SQS has the idea of a visibility timeout, where receiving and deleting the message are two different operations.

Sequence:

1. Consumer receives next message from queue. This makes the message invisible to other consumers for the set visibility timeout. Any receive calls will receive other/no messages.
2. Consumer attempts to process the message.
3. If the message is processed successfully, consumer deletes it from queue. (Last step for consumer)
4. If the consumer fails to process the message, crashes etc. it will not get as far as deleting the message from the queue. The message will then re-appear for consumption.

```
# receive message
$Message=$(aws sqs receive-message `
--queue-url $QueueUrl
| ConvertFrom-Json).Messages

# process the message

# then delete using its receipt handle
aws sqs delete-message --queue-url $QueueUrl --receipt-handle $Message.ReceiptHandle
```

Of course, can avoid this behaviour by just deleting message immediately on receipt.

The default visibility timeout is 30 seconds. Can be set for queue by modifying queue attributes:

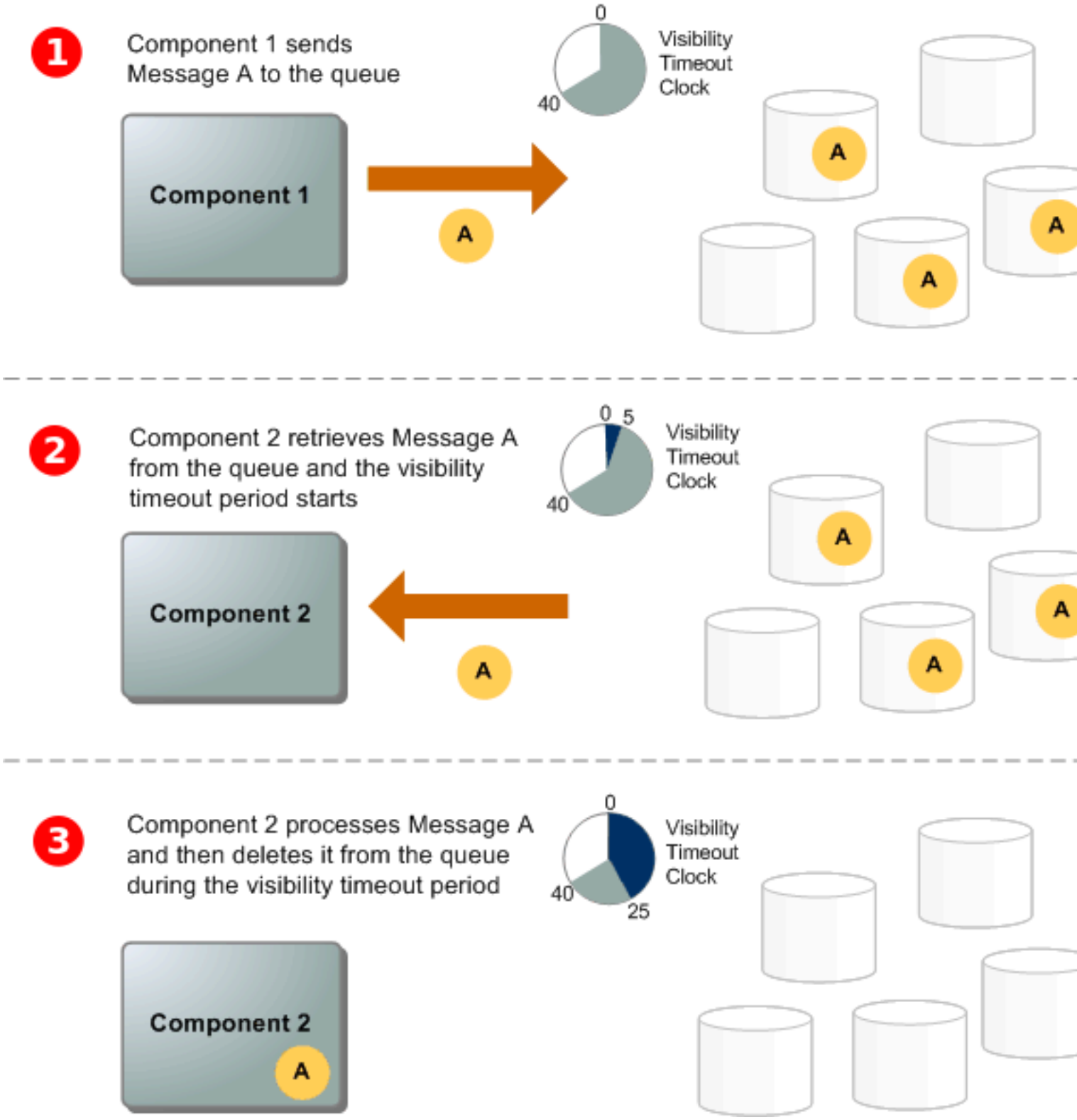

Figure 3: Consumer processing message with Visiblity timeout

```
aws sqs set-queue-attributes `
--queue-url $QueueUrl `
--attributes VisibilityTimeout=3600
```

2.5 Retention period

Messages remaining in the queue longer than the Retention Period are automatically deleted.

```
# set retention period to 1 hour (=60*60 seconds)
aws sqs set-queue-attributes `
--queueurl $QueueUrl `
--attributes MessageRetentionPeriod=3600
```

2.6 Introduce a delay

Messages can be delayed from being received after being sent to queue:

```
# keep messages back for 60 seconds
aws sqs set-queue-attributes `
--queueurl $QueueUrl `
--attributes DelaySeconds=60
```

2.7 Purging a queue

Purging a queue removes all messages from it.

```
aws sqs purge-queue --queue-url $QueueUrl
```

3 Exercise

1. Create a queue called labq.
2. Use PowerShell to construct a producer and consumer. The consumer should take some action on the message (e.g. print to screen, write to file). Build a time-delay into the consumer to simulate processing.
3. Create an S3 bucket. Modify your consumer to put each message as a new object into the S3 bucket. Write a script that loops over the S3 bucket and prints the messages.
4. Optional: replace / augment the PowerShell producer or consumer with an equivalent in another language.