Message Queues

Overview

- Message Queues
- Event Driven Design
- Pub/Sub Model and Notifications

Learning Objectives

- Understand what a queue is, and its use cases
- Understand how system design can change to utilise queues
- Understand the pub/sub model, and its use cases
- Create/use a queue and a notification topic in AWS

What is a message?

- The data transported between the sender and the receiver application. It could be:
  - A binary blob
  - Encoded data (e.g. JSON/XML etc.)
  - Could include different attributes (key/values)

'Message' is a broad definition - it can refer generally to any data sent between a sending and receiving service (or a 'producing' and 'consuming' service).

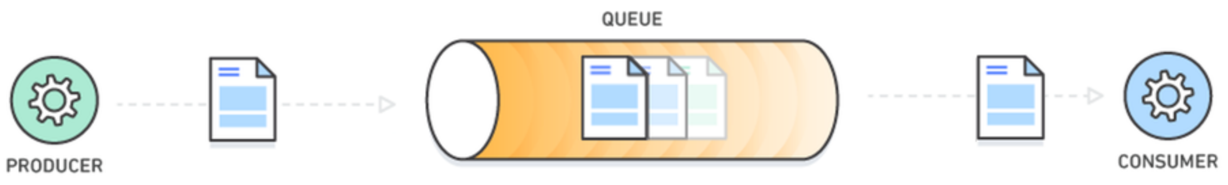
What is a message queue?

Conceptually you can think of them in the same way as a physical queue where in most cases the items in the queue are processed in the order they joined.

A producing service will send a message to a queue, where it will 'wait' to be consumed by a consuming service. Additional messages may be sent while the first message is still waiting, and they will queue up behind it. Usually, once the first message has been consumed, it will be removed from the queue, and the next message moves to the 'front' of the queue to be consumed.

Producer and Consumer Pattern

A Producer creates a message and puts it on a queue



...and a Consumer takes messages off the queue to process

Messages are stored on the queue until they are processed and deleted. Usually message should be processed only once to maintain data integrity.

Why do we use them?

We now know what a message and a queue are, but why are they useful?

- Indirect one way communication
- Process-intensive applications can be decoupled to prevent impact on other services
- Easier to replace services without changing dependent services

They provide an indirect one-way communication channel between the Consumer and Producer. This can be especially useful for decoupling heavyweight processing applications to prevent them impacting other applications in the system.

This makes it much easier to totally replace components in a system without having to amend its dependencies.

Quiz Time! 🤓

Which of the following would be a valid message to send to a queue?

1. `"I am a message!"`
2. `{"date": "01/01/2021", "content": "I am a message!"}`
3. `11011000 10101101 10001101 100110001`
4. `All of the above`

Answer: `4`

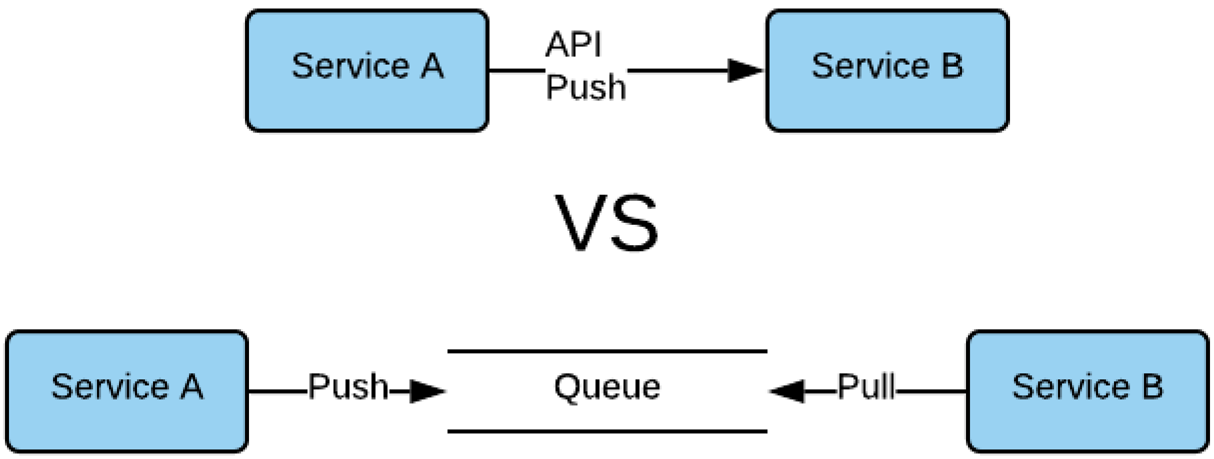
A message producer will...

1. Send a message to a queue
2. Take a message from a queue
3. Both of the above
4. Neither of the above

Answer: `1`

Service Decoupling

Service A does not need to know anything about Service B and likewise for Service B



Decoupled service-to-service communication makes it simpler to replace components without requiring changes to other components.

If Service A pushes to Service B directly - it needs to route to the correct endpoint, use proper protocols, headers etc.

Where we use a queue instead - Service A can send a message to the queue without having to know about these things. Service B can also consume the message from the queue without having any real knowledge of Service A.

In practice, it makes sense to maintain the contract between services even with a queue in the middle. If Service A changes the structure of the messages it sends to the queue, Service B will still consume these, but may not be able to process them. So the consuming Service should have some knowledge of the message format being sent by the producing service. Testing can help to make sure both stay in sync.

This design also makes it much easier to replace components - you could replace Service A with a completely new service that sends a similarly formatted message to the queue, and Servie B would never even notice.

Let's create a queue

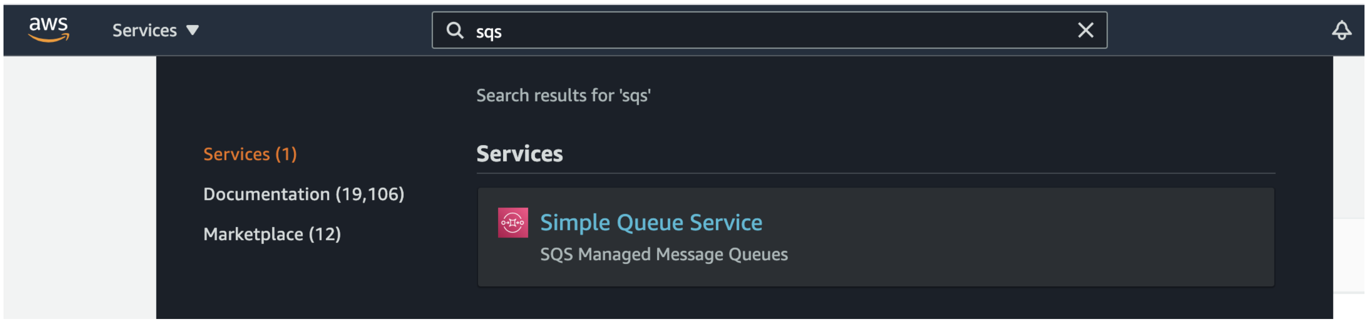
- We'll use AWS Simple Queue Service (SQS)
- We can create a queue using the AWS console
- We'll put an item onto the queue
- And we'll have a look at the items on the queue

DEMO

How to create an SQS queue

Go to SQS in AWS

- Login to AWS
- Click on "Services" and search for "SQS"
- ...select Simple Queue Service and then `Create Queue`



Search results for 'sqs'

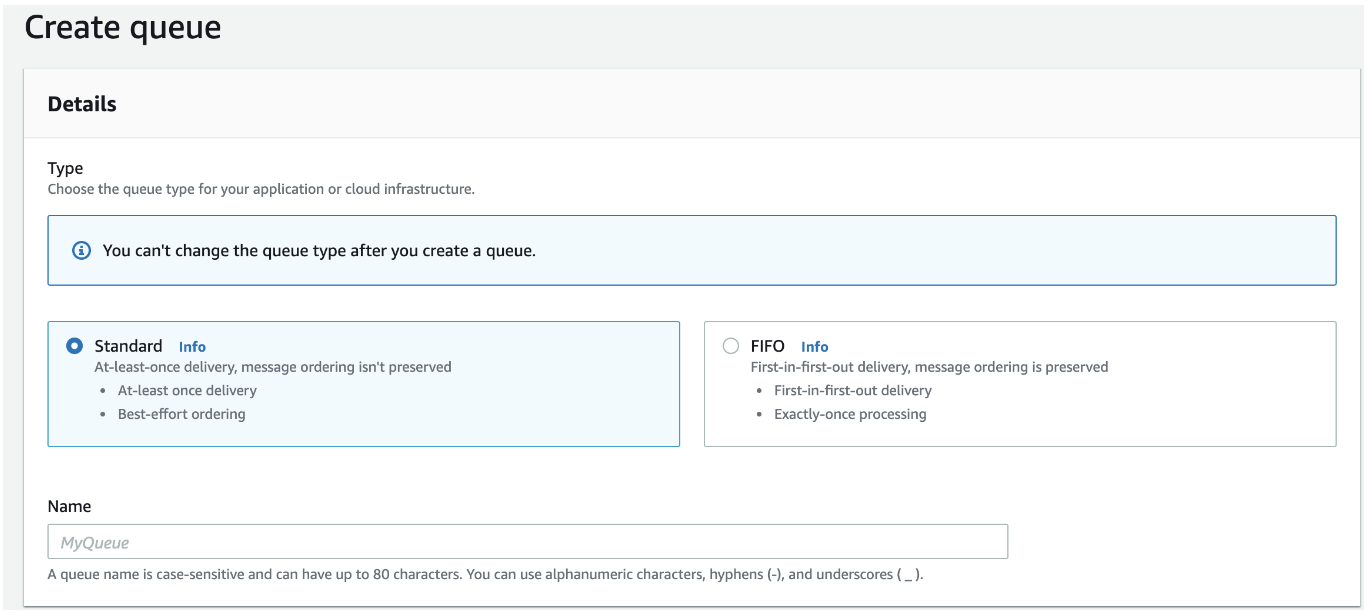
Services (1)
Documentation (19,106)
Marketplace (12)

Services

Simple Queue Service
SQS Managed Message Queues

Create a queue

- Give your queue a name (prepend with your username)
- We'll just go with a "Standard Queue"
- Accept the other standard configuration and select `Create Queue` at the bottom of the screen



Create queue

Details

Type
Choose the queue type for your application or cloud infrastructure.

You can't change the queue type after you create a queue.

[✓ Standard] Info
At-least-once delivery, message ordering isn't preserved
- At-least once delivery
- Best-effort ordering

FIFO Info
First-in-first-out delivery, message ordering is preserved
- First-in-first-out delivery
- Exactly-once processing

Name
MyQueue
A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (_).

Let's put a message on the queue

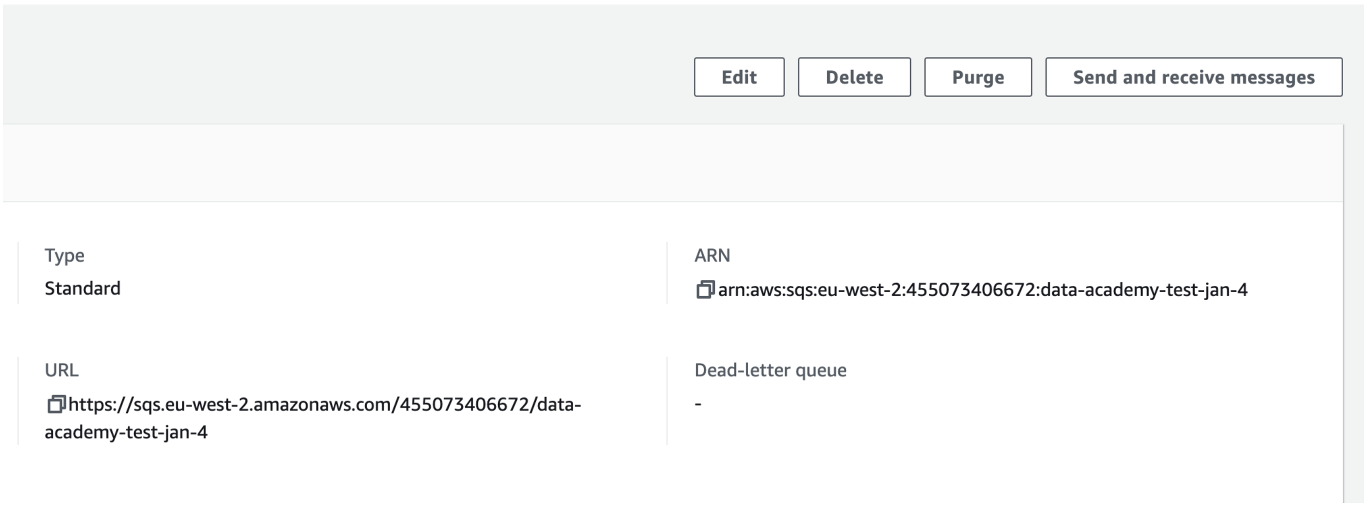
- Get the URL of your queue (it will be on the screen after you create the queue)
- We can submit a message using the AWS CLI

```
aws sqs send-message --queue-url <insert-queue-url-here> --message-body "An important message"
```

We can now inspect the contents of the queue

- Go back to the SQS console

- It should display 1 "Messages Available" next to your queue
- Click on the queue name and select "Send and receive messages"



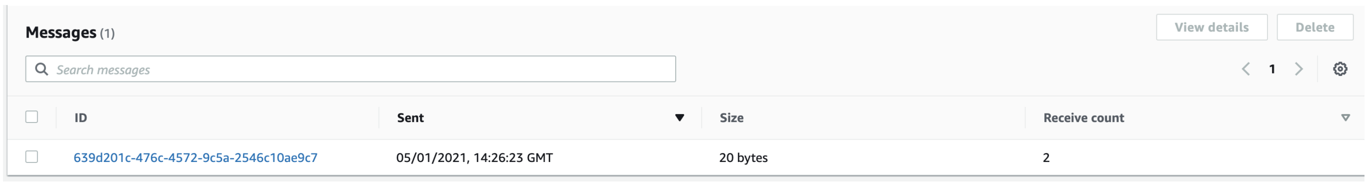
Type
Standard

ARN
arn:aws:sqs:eu-west-2:455073406672:data-academy-test-jan-4

URL
https://sqs.eu-west-2.amazonaws.com/455073406672/data-academy-test-jan-4

Dead-letter queue
-

- You may have to scroll to the bottom and click 'Poll for Messages', then it should appear in the list of messages
- Click on the message to see the content of the message you sent



ID	Sent	Size	Receive count
639d201c-476c-4572-9c5a-2546c10ae9c7	05/01/2021, 14:26:23 GMT	20 bytes	2

Event driven design

- Be told when something happens rather than asking
- This is more efficient than polling for changes
- Queues can form a key component of an event driven architecture
- The publisher/subscriber (pub/sub) design is also key

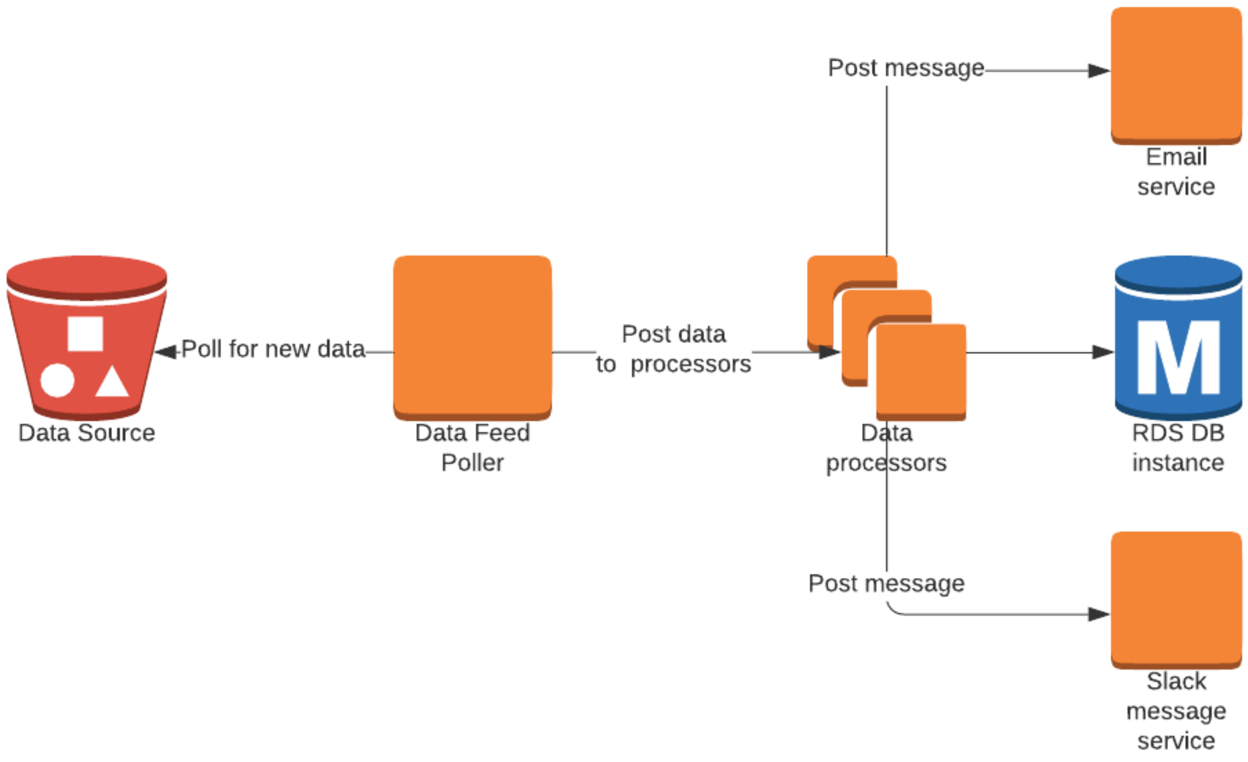
Event driven design is a relatively modern idea on how to design systems. The events that are published and consumed, processed and persisted, are the core of the system. This differs from the generally prevailing pattern of systems being driven by requests between different components.

Such a system will be heavily decoupled by design - none of the systems that publish messages will know or care what consumes them, and events themselves aren't aware of the consequences of their processing.

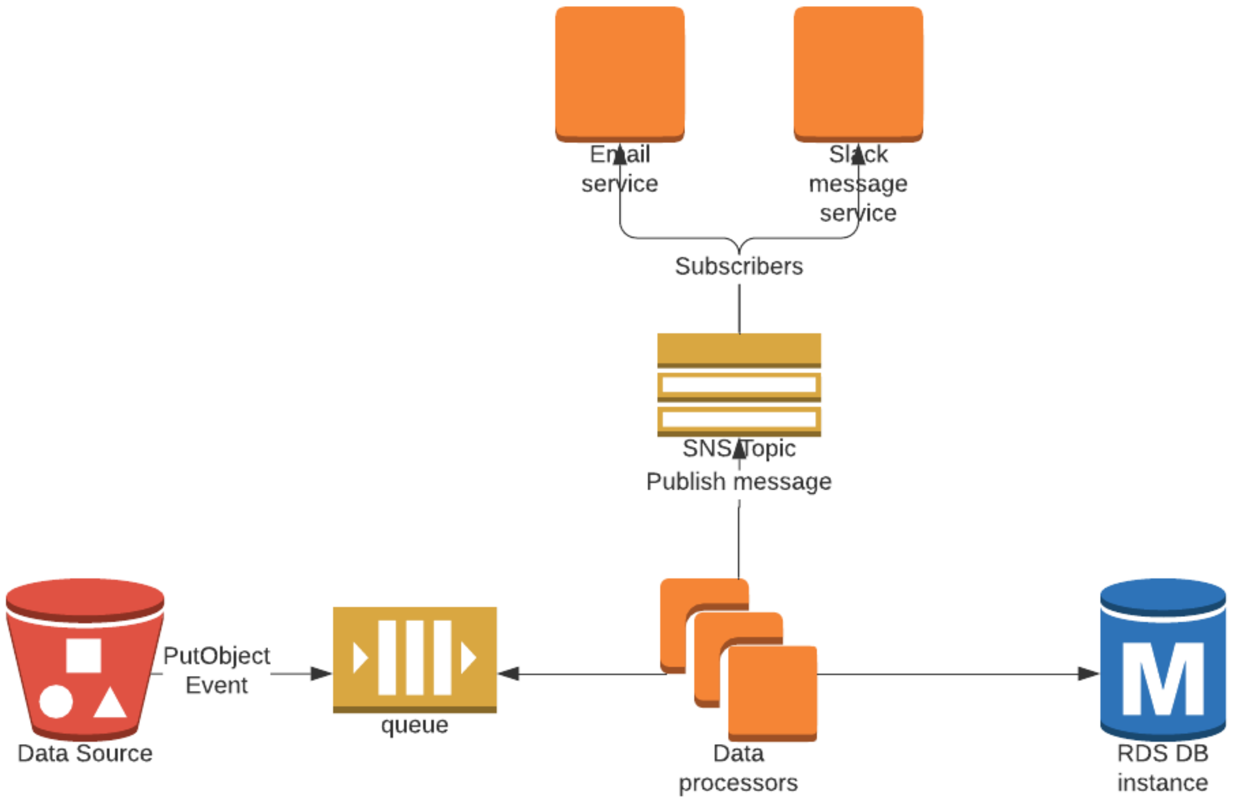
Other benefits include scalability (you can always add new components which work with existing events), adaptability (quite easy to independently replace components) and availability of 'real-time' data.

Potential drawbacks include difficulty to monitor - when you don't have direct flows through the system, it can be hard to follow!

Non event driven



Event driven



What's the difference?

- We are notified when there is new data instead of checking all the time
- The data processor is now in control of how much data it processes
- We can add new subscribers to events dispatched from the data processor without changing the data processor

---

Fault Tolerance

- The ability to gracefully degrade during failures in a system
- If the Consumer is unavailable, messages are not lost
- Messages can be processed once the Consumer is available again

Being able to survive and recover from failures of components is an important factor of robust systems. If the Consumer is unavailable messages from the Producer will not be lost. They will remain on the queue until such time the Consumer is in a position again to process them.

Traffic Smoothing

In situations where you can have sudden spikes in load to an application, using a queue you can control the rate of data processing to prevent the application becoming overloaded.

Granular Scaling

PRODUCER
CONSUMER

Depending on the demands on the system, you can scale the number of Consumers and Producers independently. These can grow and shrink as the workload requires.

Handling failures

What happens if the message cannot be processed?

- Retry n times to process, there could be a temporary issue
- Configure a visibility timeout - how long to wait for a Consumer before making it visible on the queue again
- Configure a redrive policy - how many processing attempts per message
- Send it to a Dead Letter Queue (DLQ)

Dead Letter Queues

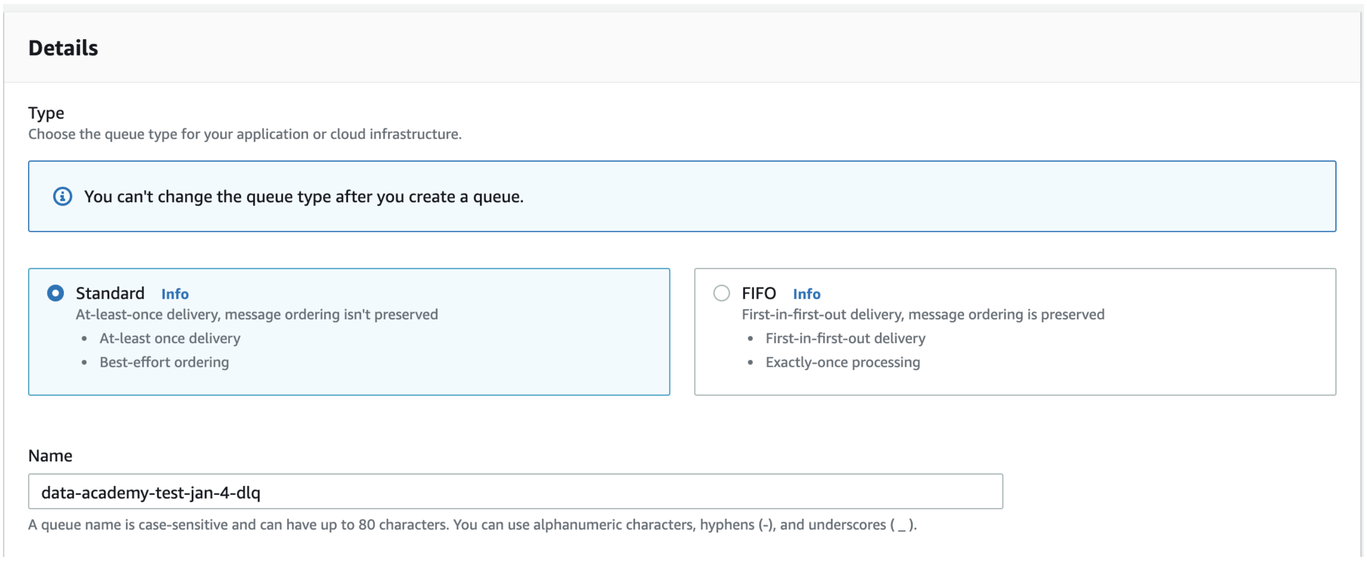
Another queue where you can send messages that could not be processed. Useful for retaining the messages for inspection without them continuing to be processed by the consumer.

DEMO

How do we set up dead letter queues?

Create Dead Letter Queue

- We follow the same method as we did to create a queue above
- Use the same name as beofre, but append with `-dlq`



Details

Type
Choose the queue type for your application or cloud infrastructure.

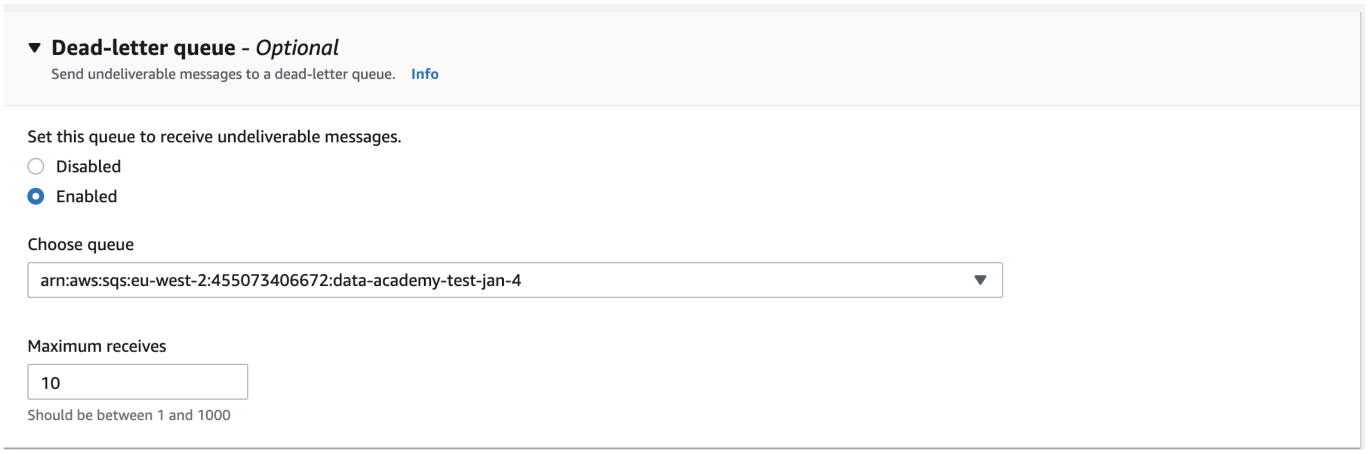
You can't change the queue type after you create a queue.

[✓ Standard] Info
At-least-once delivery, message ordering isn't preserved
- At-least once delivery
- Best-effort ordering

FIFO Info
First-in-first-out delivery, message ordering is preserved
- First-in-first-out delivery
- Exactly-once processing

Name
data-academy-test-jan-4-dlq
A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (_).

- On the same screen, one of the optional features is 'Dead-letter-queue'. This time, set it to `Enabled` and choose the queue created earlier in the dropdown list, then click `Create queue`
- This queue will now receive messages which could not be consumed by the earlier queue



Dead-letter queue - *Optional*
Send undeliverable messages to a dead-letter queue. Info

Set this queue to receive undeliverable messages.
Disabled [✓ Enabled]

Choose queue
arn:aws:sqs:eu-west-2:455073406672:data-academy-test-jan-4

Maximum receives
10
Should be between 1 and 1000

Features of queues

- Different queue solutions offer different features

- Use case may dictate technology choice

Push or pull delivery

Pull means continuously querying the queue for new messages. Push means that a consumer is notified when a message is available, this is also known as Pub/Sub messaging.

At least once delivery

Stores multiple copies of messages for redundancy and high availability. Messages are re-sent in the event of errors to ensure they are delivered at least once.

Exactly once delivery

When duplicates can't be tolerated, FIFO (first-in-first-out) message queues will make sure that each message is delivered exactly once.

Queue vs Pub/Sub

Amazon Simple Notification Service (SNS)

SQS is great, but what about when you need more complete patterns such as:

- Many to many messaging
- Selective routing
- Routing to multiple services

SNS Topics

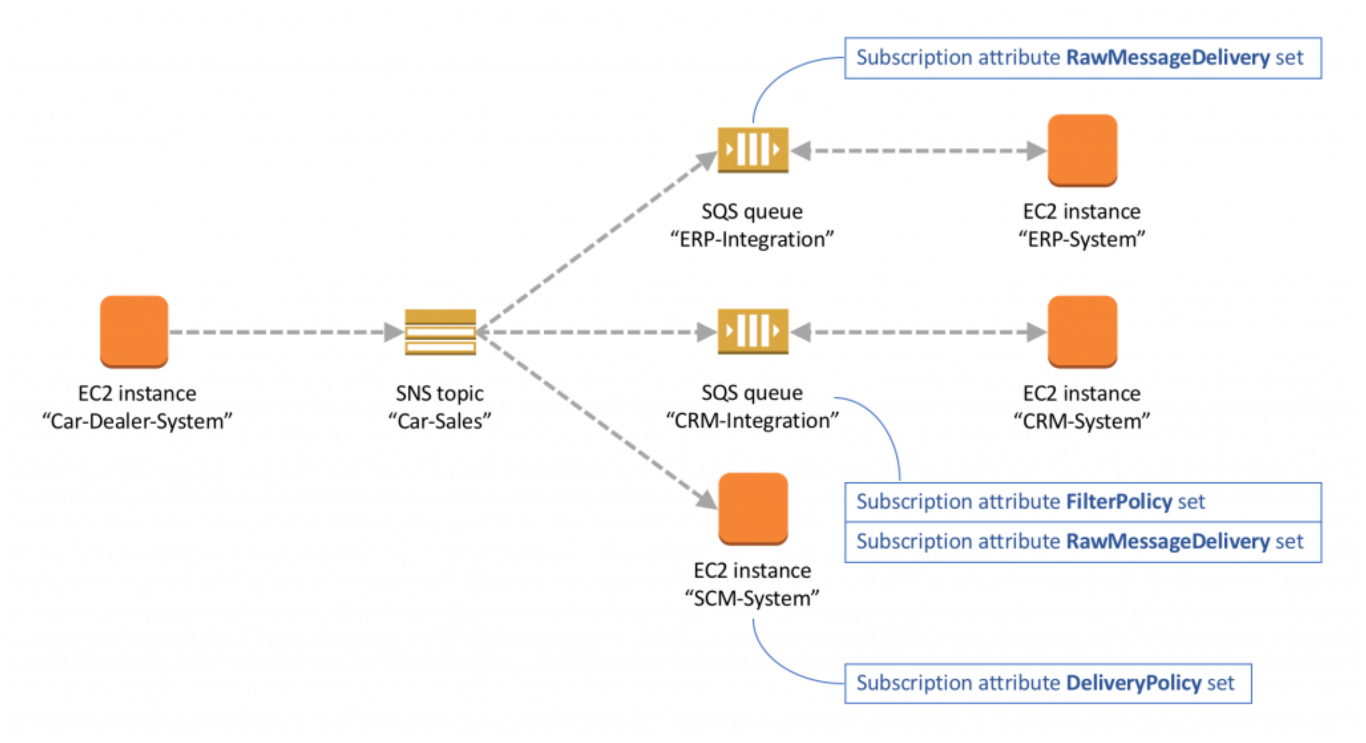
SNS introduces the concept of "topics" to solves these problems.

Producers **Publish** to a topic

Consumers **Subscribe** to that topic

Hence the name **pub/sub**

Messages sent to the topic can be subscribed to by multiple consumers.


Subscription attribute RawMessageDelivery set
SQS queue "ERP-Integration"
EC2 instance "ERP-System"
EC2 instance "Car-Dealer-System"
SNS topic "Car-Sales"
SQS queue "CRM-Integration"
EC2 instance "CRM-System"
Subscription attribute FilterPolicy set
Subscription attribute RawMessageDelivery set
EC2 instance "SCM-System"
Subscription attribute DeliveryPolicy set

We can start to see here how using the pub-sub design pattern opens up a huge amount of possibility for system design. We could publish a notification to a topic, which has various subscribers which all do different things with the message, and could have other subscribers which contribute to things like record keeping, monitoring, visualisation, notification etc.

Consumers can create subscriptions using a number of protocols:

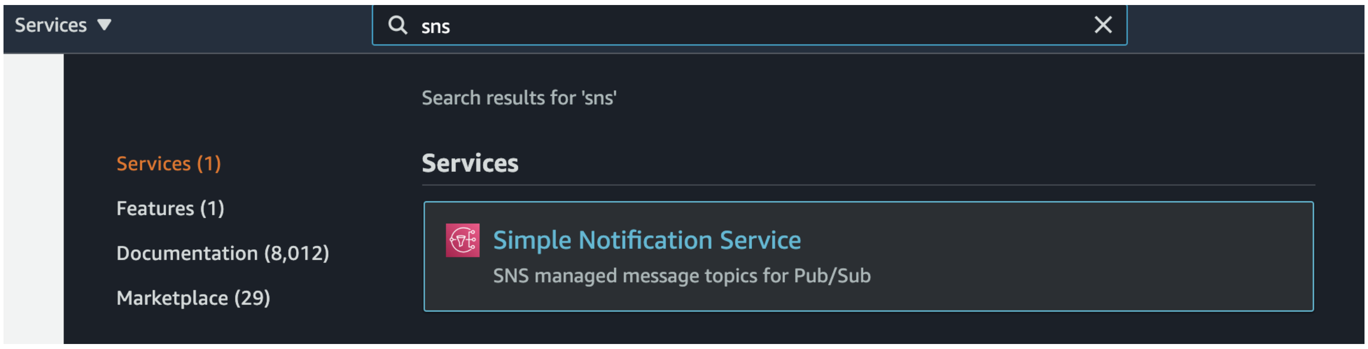
- SQS
- AWS Lambda
- HTTP/S webhooks
- Email

SNS DEMO

A walkthrough setting up an SNS topic and pub/sub behaviour.

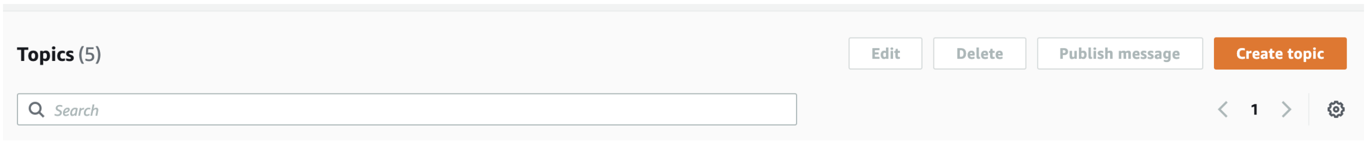
Go to SNS in AWS

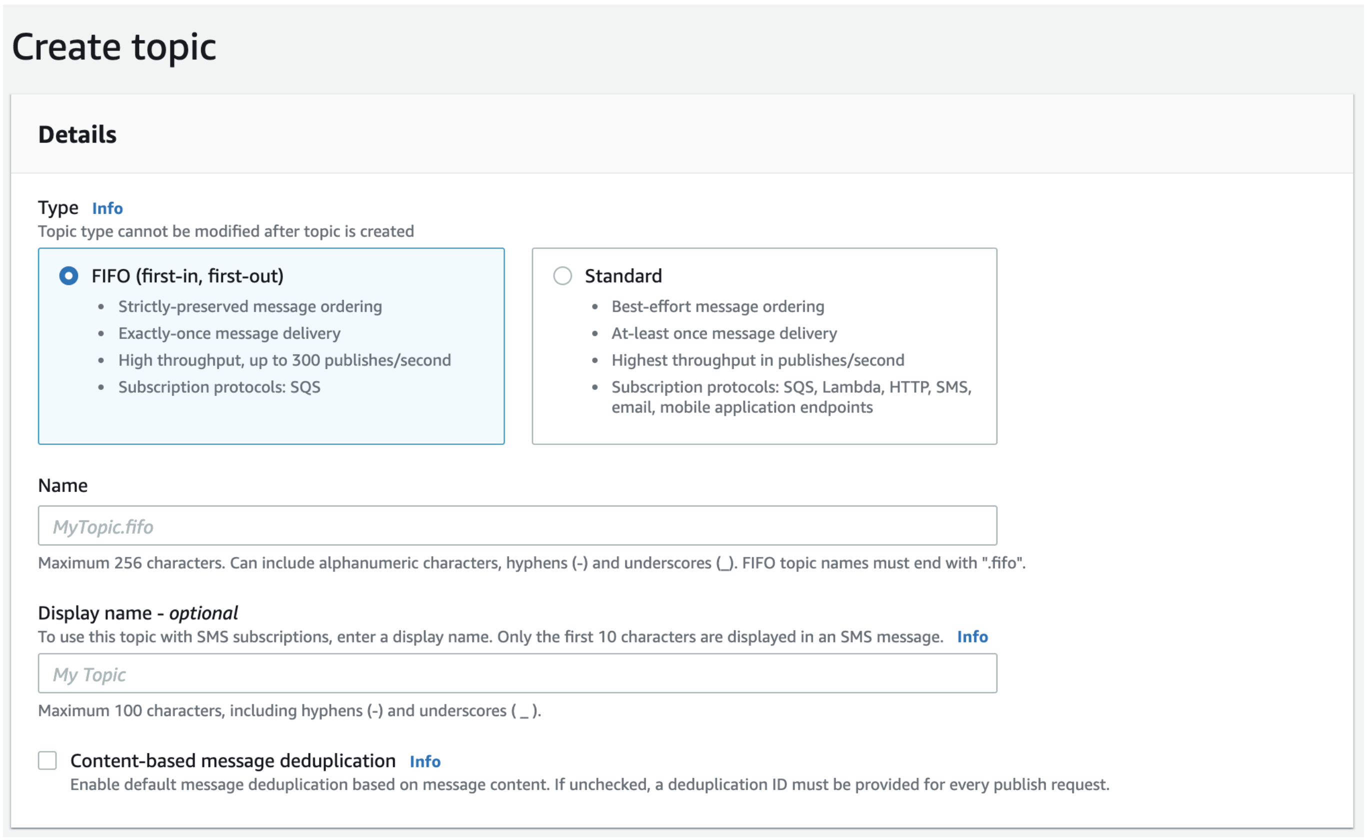
- Login to AWS.
- Click on "Services" and search for "SNS".
- ...and select Simple Notification Service.



Create a Topic

- Navigate to "Topics".
- Select "Create Topic".
- Enter a name (prepend with your username), click "Create Topic".





Create topic

Details

Type Info
Topic type cannot be modified after topic is created

● FIFO (first-in, first-out)
- Strictly-preserved message ordering
- Exactly-once message delivery
- High throughput, up to 300 publishes/second
- Subscription protocols: SQS

○ Standard
- Best-effort message ordering
- At-least once message delivery
- Highest throughput in publishes/second
- Subscription protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints

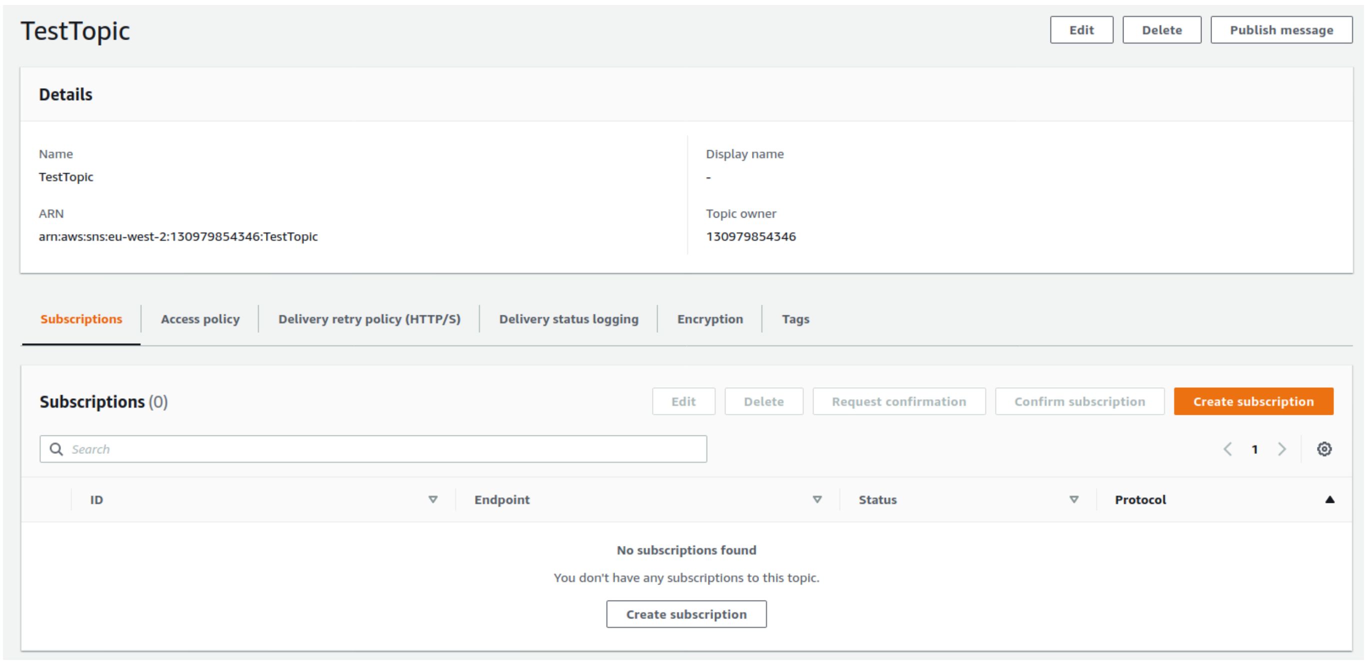
Name
MyTopic.fifo
Maximum 256 characters. Can include alphanumeric characters, hyphens (-) and underscores (_). FIFO topic names must end with ".fifo".

Display name - *optional*
To use this topic with SMS subscriptions, enter a display name. Only the first 10 characters are displayed in an SMS message. Info
My Topic
Maximum 100 characters, including hyphens (-) and underscores (_).

☐ Content-based message deduplication Info
Enable default message deduplication based on message content. If unchecked, a deduplication ID must be provided for every publish request.

Create a Subscription

- You should now see the topic summary.
- Click "Create Subscription".


TestTopic

Details

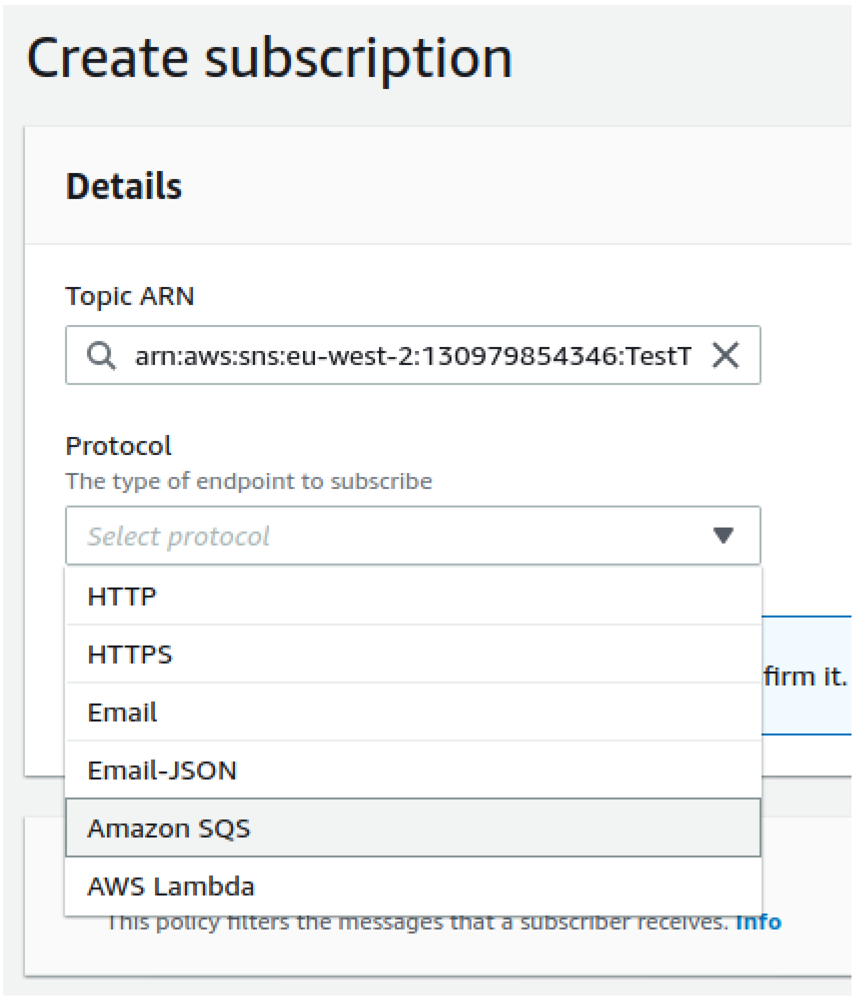
Name: TestTopic	Display name: -
ARN: arn:aws:sns:eu-west-2:130979854346:TestTopic	Topic owner: 130979854346

No subscriptions found

You don't have any subscriptions to this topic.

Create a Subscription

- The topic name should be pre-populated.
- Choose "Amazon SQS" as the protocol.
- Enter the ARN of the SQS queue that you created earlier.
- Click "Create Subscription".


Create subscription

Details

Topic ARN: arn:aws:sns:eu-west-2:130979854346:TestT

Protocol: The type of endpoint to subscribe

This policy filters the messages that a subscriber receives. Info

---

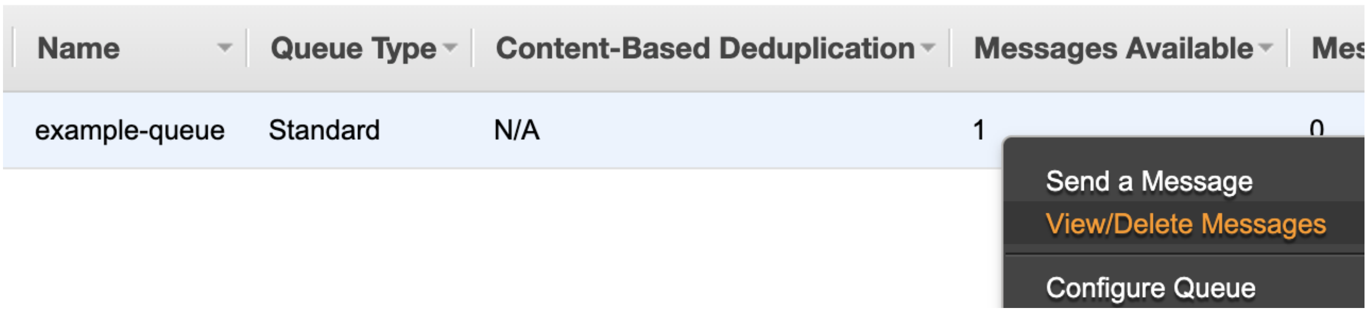
Let's put a message to the topic and read it from the queue

- Get the ARN of your topic.
- We can submit a message using the AWS CLI using, for example, `aws sns publish --topic-arn arn:aws:sns:eu-west-2:130979854346:TestTopic:7e9c2c7e-42fa-4186-9df0-700e98eb9940 --message "A pub sub message!"`
- You may need to add the "SendMessage" action to the access policy for the SQS queue.

Can this be done entirely with sqs:* IAM permissions? Will they need special IAM perms to grant permissions like this?

Once again can now inspect the contents of the queue

- Go back to the SQS console.
- It should display 1 "Messages Available" next to your queue.
- Click on the queue name and select "View/Delete Messages".
- You should be able to see the message you sent.



Name	Queue Type	Content-Based Deduplication	Messages Available	Mes
example-queue	Standard	N/A	1	0

Going forward

By attaching other subscribers to this topic (possibly using different protocols) you can share messages out to multiple consumers.

Things to be aware of

Added complexity

Using queues is more complex than just using a HTTP request to talk directly to the message consumer.

You need to determine on a case by case basis whether the additional complexity incurred is worthwhile.

Duplicate Processing

- What will happen if a message is processed multiple times
- What scenarios could cause this to happen?

Duplicate messages could occur in cases of, for example, where a temporary issue prevents amessage being properly accepted onto an SQS queue, and if there is a retry policy set then a second message is later sent. Then, for whatever, reason, the original message is successfully accepted.

Touch briefly on idea of idempotency - certain requests are idempotent in that having them execute multiple times has the same reults, while some do not (consider a GEt vs POST request). If a message is processesd multiple times and the outcome is a non-idempotent operation, we may get into trouble. You can design your system to handle this, with various duplicate checking controls around non-idempotent actions, though this can be complex and requires a lot of foresight.

Amazon SQS can help to prevent this to an extent - using deduplication ids SQS will prevent duplicate messages within a few minutes of each other.

It is best to accept that, at some point, duplicate messages will be encountered, and plan on how to deal with this.

Message Throughput

- Data messages are restricted in size (SQS messages max size is 256kb)
- If data is larger than this consider sending a reference to the file
- You may be limited to a maximum number of messages per second you can process

Depending on the message queue you are using, and the configuration of it, you may be constrained by a number of messages per second (MPS) you can add to the queue and also by a maximum individual message size.

Learning Objectives Revisited

- Understand what a queue is, and its use cases
- Understand how system design can change to utilise queues
- Understand the pub/sub model, and its use cases
- Create/use a queue and a notification topic in AWS

Terms and Definitions Recap

Message: A broad term for data passed between services.

Queue: A method of service-to-service communication where the the sender and receiver of a message don't interact at the same time. Messages placed on a queue stay there until consumed.

Producer: Service sending a message to a queue

Consumer: service taking a message from a queue to process

Terms and Definitions Recap

Decoupling: removing dependencies between services, e.g. by using a queue

Fault Tolerance: A system's ability to gracefully handle and recover from failure of a component

Dead Letter Queue: A queue where you can send messages that couldn't be processed, to be handled at a later time

Pub/Sub: a design pattern where messages are sent to a topic, and numerous services can subscribe to this topic

---

Further Reading

Intro to message queues

Amazon SQS

Intro to Event Driven Architecture

Intro to the pub/sub design pattern

Amazon SNS