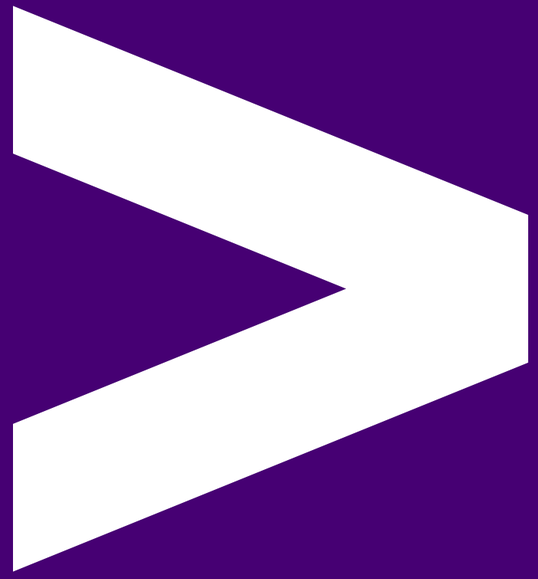
School of Tech
part of accenture

Message Queues with AWS



School of Tech
part of accenture

AWS sessions list

This is the list of AWS sessions done so far, and the following ones:

- AWS 01 AWS + Cloud Intro ✅ *1.5hrs*
- AWS 02 AWS CLI Setup ✅ *1.5hrs*
- AWS 03 S3 Storage (Console) ✅ *1.5hrs*
- AWS 04 CloudFormation Intro + S3 Storage (IaC) ✅ *1.5hrs*
- AWS 05 Lambda Intro ✅ *1.5hrs*
- AWS 06 Lambda (IaC) ✅ *1.5hrs*
- AWS 07 Redshift (IaC) ✅ *1.5hrs*
- AWS 08 EC2 (IaC) + Grafana setup ✅ *1.5hrs*
- AWS 09 Queues ⬅ *1.5hrs*
- AWS 10 Monitoring *1.5hrs*

School of Tech
part of accenture

Overview

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

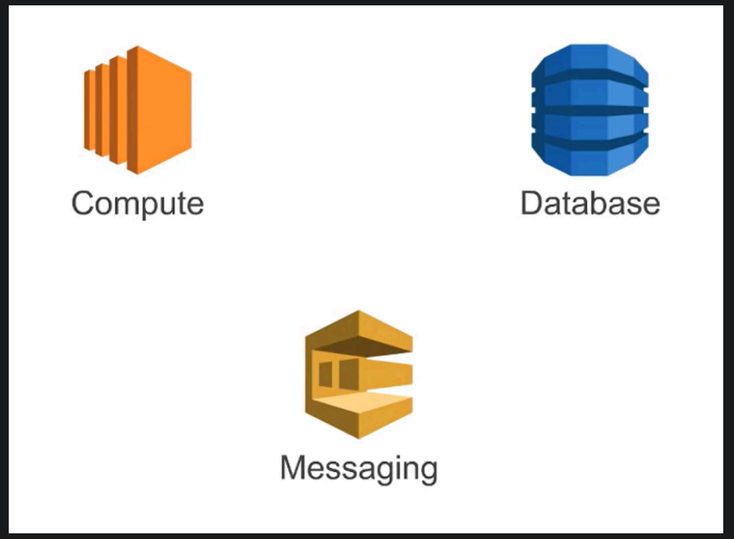
School of Tech
part of accenture

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

School of Tech
part of accenture

Modern Apps



When you are creating modern app in the cloud, there are often three main pillars:

- Compute: ec2, lambda, ...
- Database: rds, ...
- Messaging

School of Tech
part of accenture

Modern Apps

Messaging is the topic of today, and it's a kind of glue that often connects other high-performing pieces together.

When we are talking about messaging the most important thing is not just the message itself, but how it lets other components run as fast as possible by "de-coupling" them.

What is a message?

A "message" is the data transported between the sender and the receiver application. It could be:

- A binary blob
- Encoded data (e.g. JSON/XML etc.)
- Can include many different attributes (key/values)
- A link to (or name of) a file, where the file is large

School of Tech
part of accenture

Sample message JSON

Many systems use JSON formatted messages, for example maybe this for data delivery:

```
{
    "type": "sale",
    "item": "Coffee",
    "total": 1.77
}
```

or this for notification of a file we could process:

```
{
    "type": "daily-report-file",
    "location": "s3://coffee-data-bucket/sales_data.csv",
    "timestamp": "13-Jan-2025 18:00"
}
```

School of Tech
part of accenture

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 queue of people outside the original Sainsburys supermarket!

What is a message queue? (cont)

- A durable buffer for our messages
- Message queues can be used to decouple heavyweight processing, to buffer or batch work, and to smooth spiky workloads
- A message queue is a form of asynchronous service-to-service communication used in serverless and microservices architectures

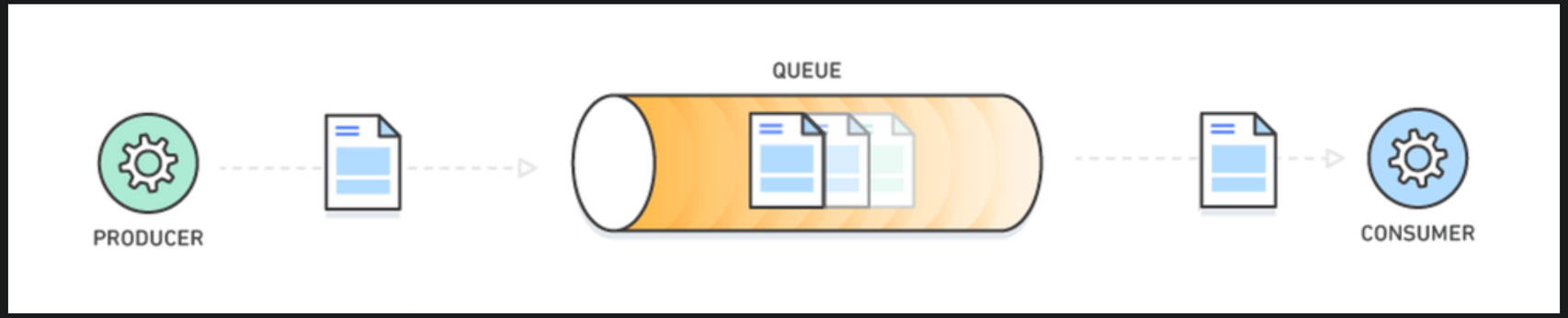
What is a message queue? (cont)

- Messages are stored on the queue until they are **processed** and **deleted**
- Each message is usually processed only once, by a single consumer
- 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.

School of Tech
part of accenture

Producer and Consumer Pattern

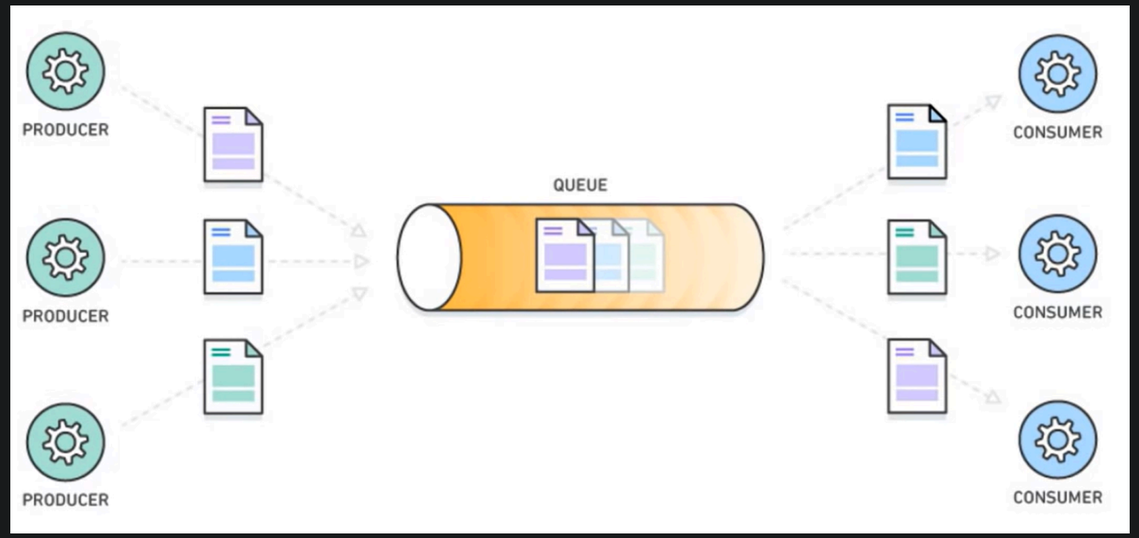
A Producer creates a message and puts it on a queue



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

School of Tech
part of accenture

Producer and Consumer Pattern



Practice (code-along)

Create a python script named `dummy_queue.py` with the following functions - keep the FIFO (First In, First Out) principle in mind:

```
queue = []

def produce(value):
    pass

def consume():
    pass
```

School of Tech
part of accenture

Practice (code-along)

Usage:

```
queue = []

def produce(message):
    queue.append(message)
    print(f"Produced message: {message}")

def consume():
    print(f"Consumed message: {message}")
    return message

produce("Sold a coffee")
produce("Sold a latte")

consume()  # "Sold a coffee"
consume()  # "Sold a latte"
consume()  # "No message to consume."
```

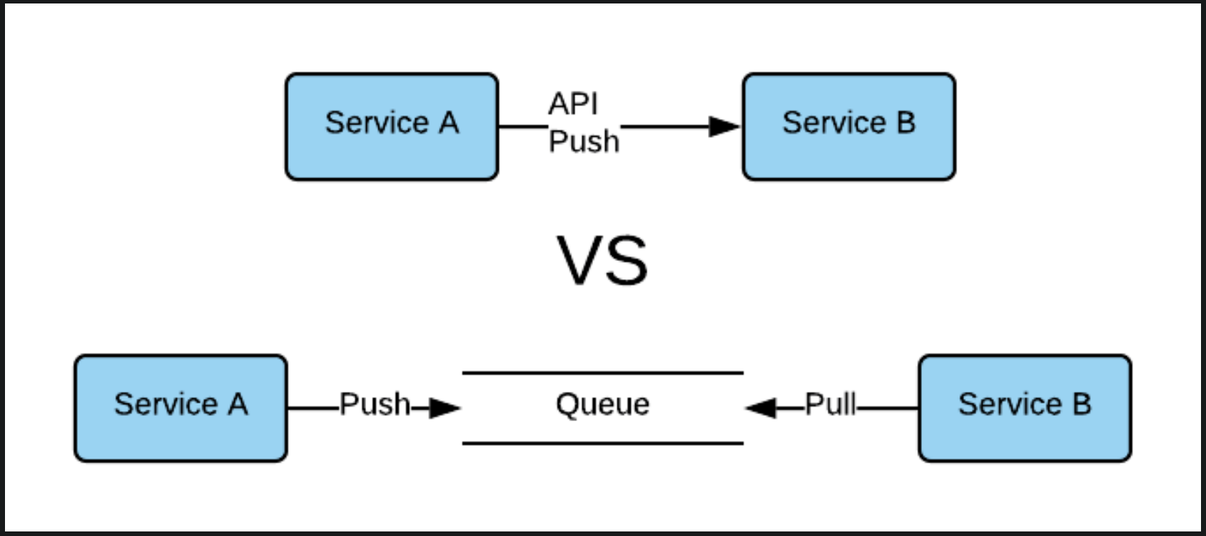
Why do we use them?

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

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.

School of Tech
part of accenture

Emoji Check:

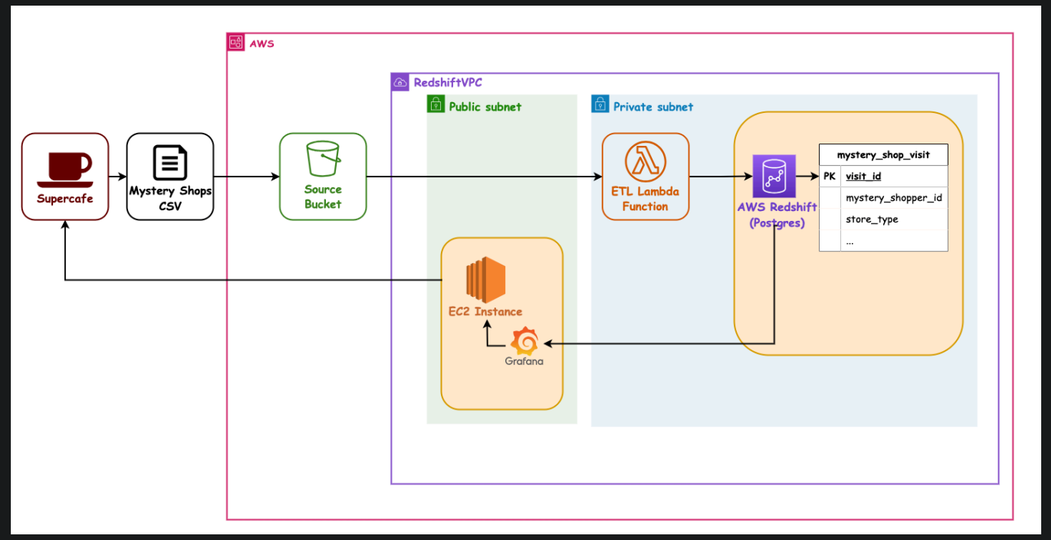
How did you find the concept of Queues?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😄 Yes, enough to start working on it collaboratively

School of Tech
part of accenture

Back to Mystery Shopper

Consider the Mystery Shopper example;



Discussion - 5 mins

> Where in the Mystery Shopper architecture might we put queues?
>
> And - why would you put them there?

School of Tech
part of accenture

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

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

SQS Demo - code along

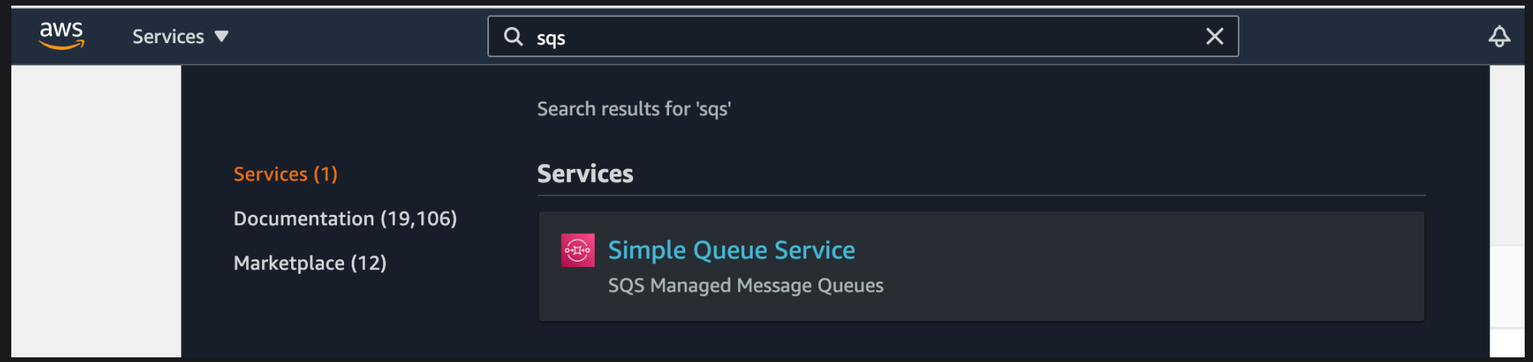
> A walkthrough setting up an SQS queue.

Steps on next slides.

School of Tech
part of accenture

Go to SQS in AWS

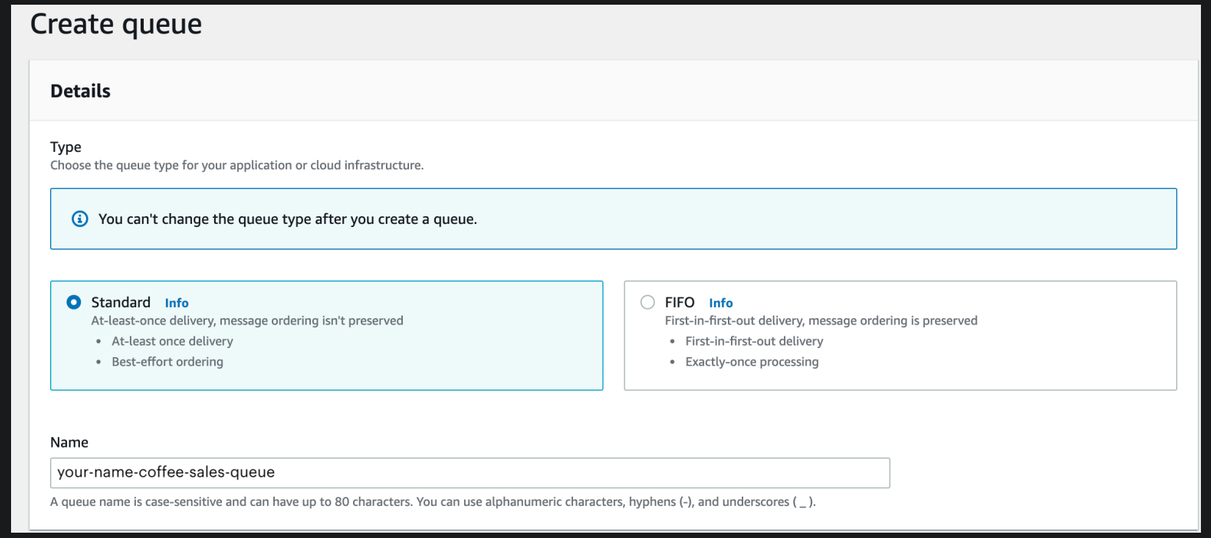
- Login to AWS
- Click on "Services" and search for "SQS"
- Select "Simple Queue Service" and then `Create Queue`



School of Tech
part of accenture

Create a queue

- Select "Standard Queue" as the type
- Give your queue a name (`<your-name>-coffee-sales-queue`)



Create a queue

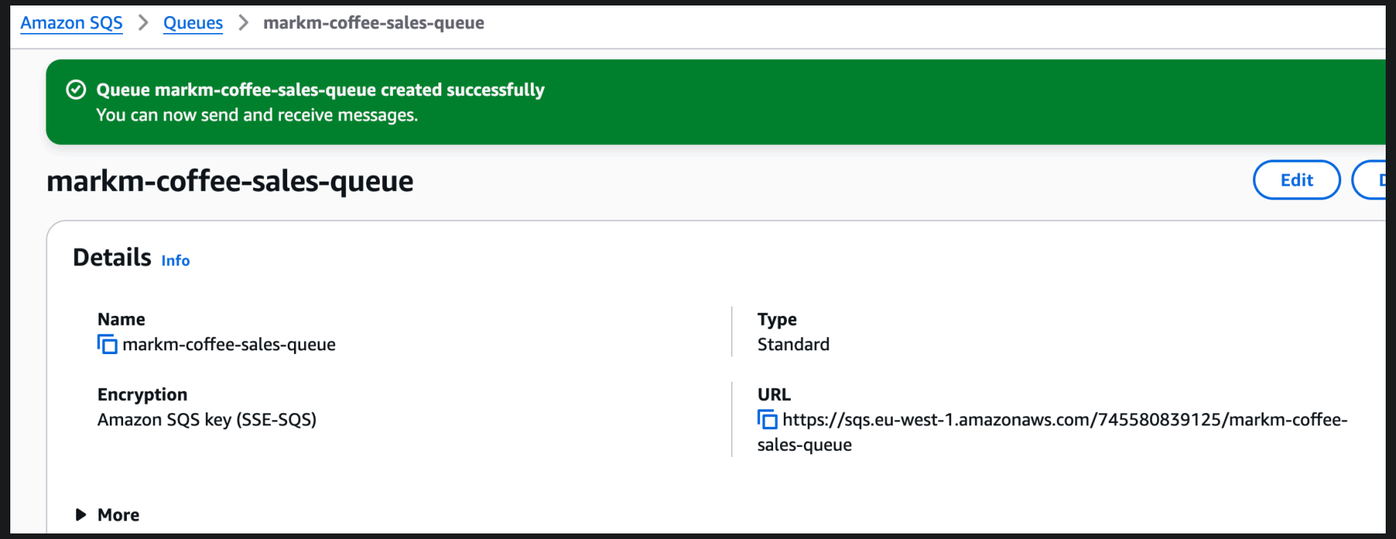
- In `Encryption`, verify `Server-side encryption` is `Enabled` (leave the other options on default)
- Leave the other standard configurations as they are
- Add a `Tag` with key `Name` and the same value as your queue name
- Finally, and select `Create Queue` at the bottom of the screen

School of Tech
part of accenture

Config for later

Now the queue is created, we need to know it's url for sending it messages.

- From the console, click the "Copy URL" button



Using AWS CLI

- Open a terminal
- Log into AWS using your standard alias or command

School of Tech
part of accenture

Let's put a message on the queue

- We can submit a message using the AWS CLI
- Substitute the URL of your queue from before

```
aws sqs send-message --queue-url <queue-url> \
    --message-body "I bought a coffee for <your-name>" \
    --profile <aws-profile>
```

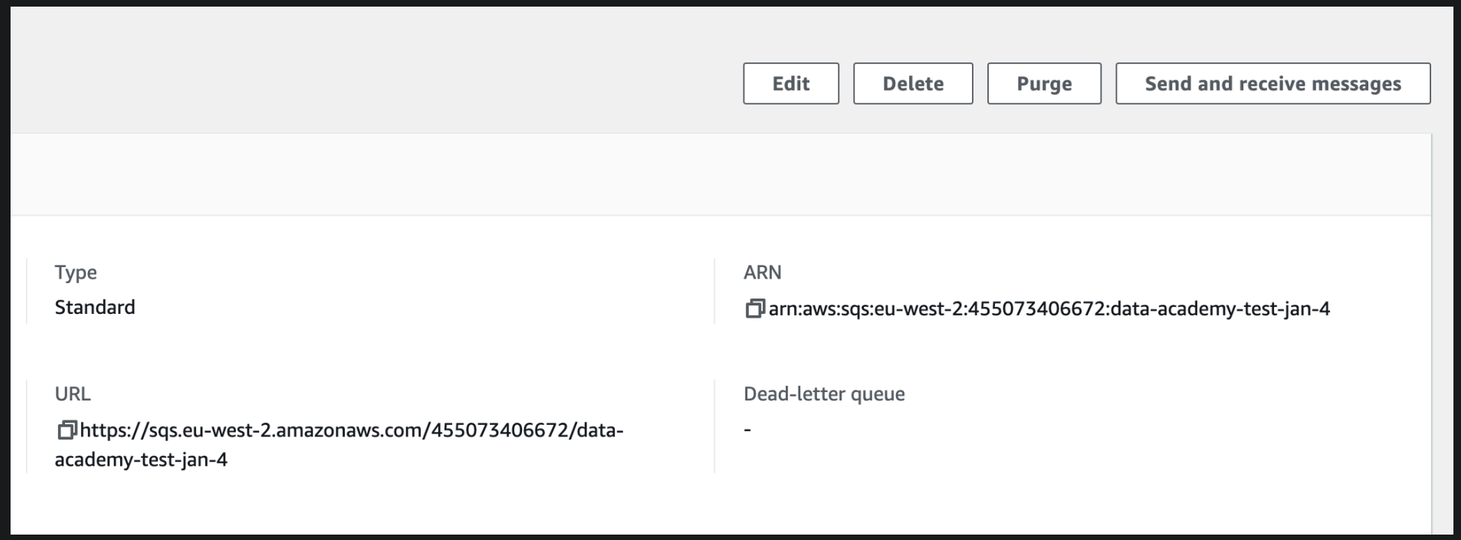
You should get a reply like

```
{
    "MD5OfMessageBody": "6a5dc09a156f2dcd3e526713093157fb",
    "MessageId": "d04b9e8d-c033-47e2-b707-f42861332ac6"
}
```

School of Tech
part of accenture

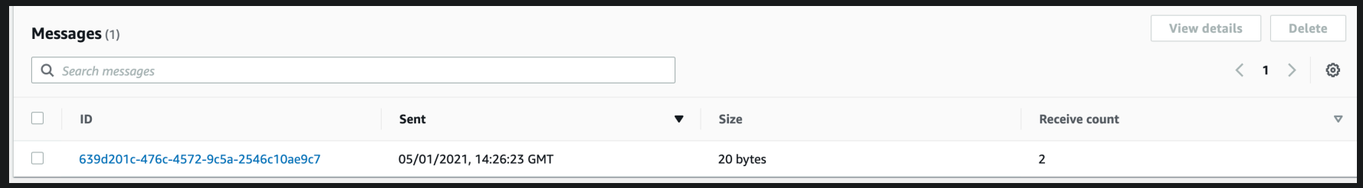
We can now inspect the contents of the queue

- Go back to the SQS console
- Refresh the master list of all queues
- It should display 1 "Messages Available" next to your queue
- Click on your queue name and select "Send and receive messages"



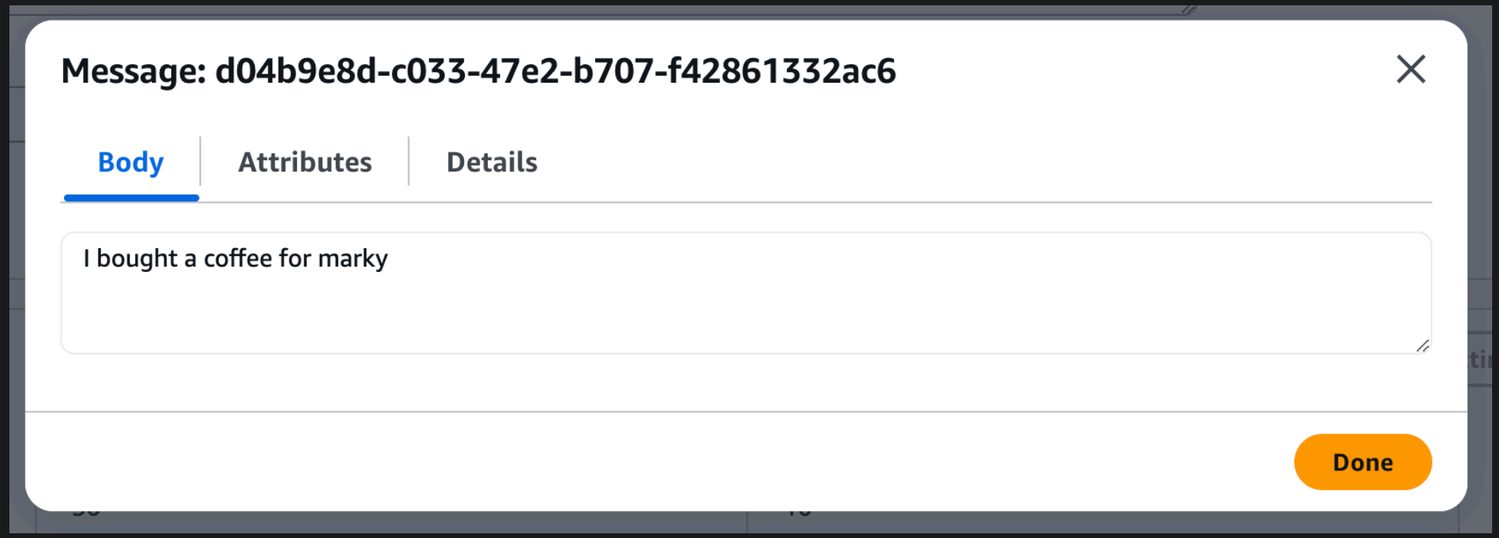
School of Tech
part of accenture

- 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 id

- You should see your text



> Awesome! we have now consumed a message

Once more with feeling

> Now we have sent a message with the CLI, lets send one with some Python code.

- Make sure you have a venv active
- Run `python3 -m pip install boto3`
    - You may need to use `python` on windows or `py` instead of `python3`

School of Tech
part of accenture

Code along - put a message on the queue using BOTO3

```
import boto3
import json

session = boto3.Session(profile_name='<aws-profile>')
sqs_client = session.client('sqs')

message = { "item": "latte", "price": 2.25 }
```

continued next slide

School of Tech
part of accenture

Code along - put a message on the queue using BOTO3

```
response = sqs_client.send_message(
    QueueUrl="<url-of-the-queue>",
    MessageAttributes={
        'Author': {
            'StringValue': '<your-name>',
            'DataType': 'String'
        }
    },
    MessageBody=json.dumps(message)
)

print(f'response = {response}')
```

Code along - check sending

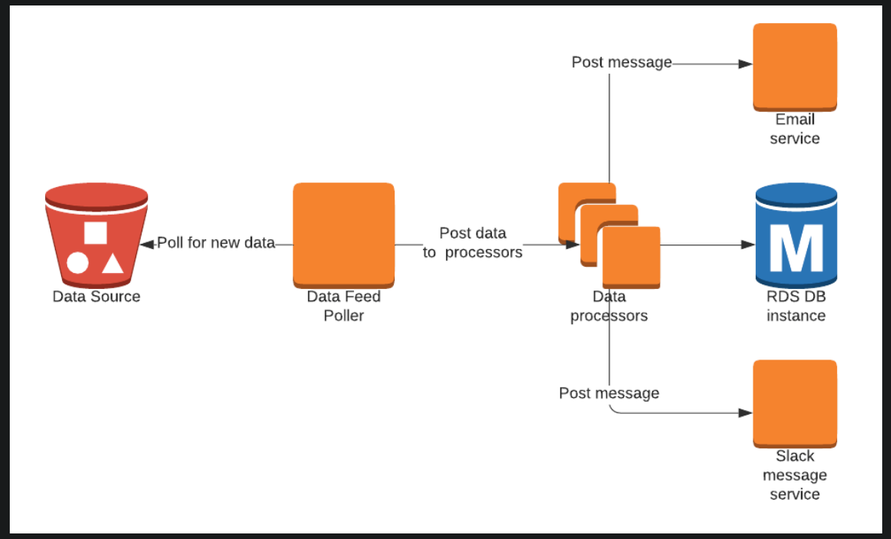
> Go back to the AWS SQS Console and check for your latest message.

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

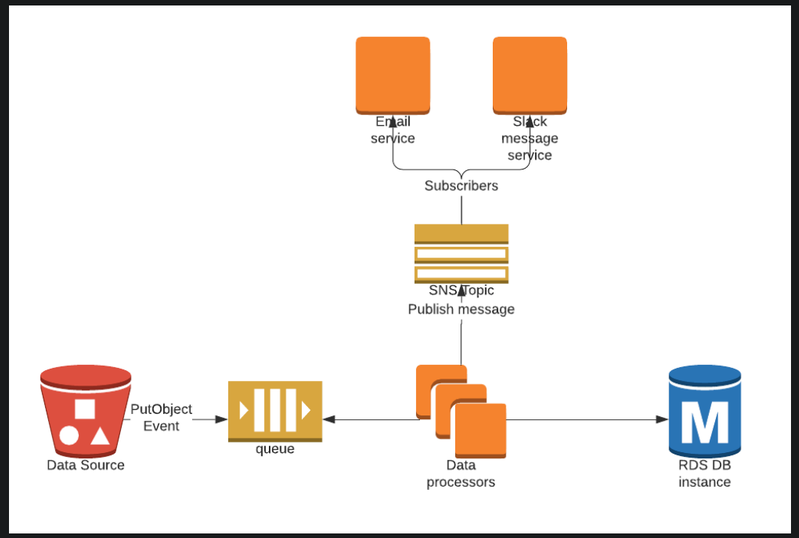
School of Tech
part of accenture

Non event driven



School of Tech
part of accenture

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 immediately lost
- Messages can be processed once the Consumer is available again

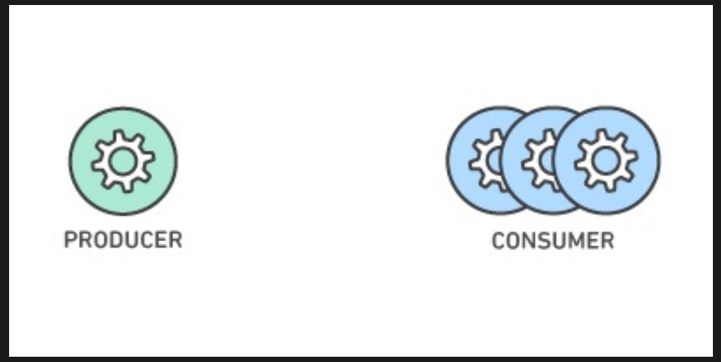
School of Tech
part of accenture

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.

School of Tech
part of accenture

Granular Scaling



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.

School of Tech
part of accenture

Emoji Check:

How did you find the implementation of AWS SQS queues and it's implementation for event-driven system design?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

School of Tech
part of accenture

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.

DLQ Demo - code along

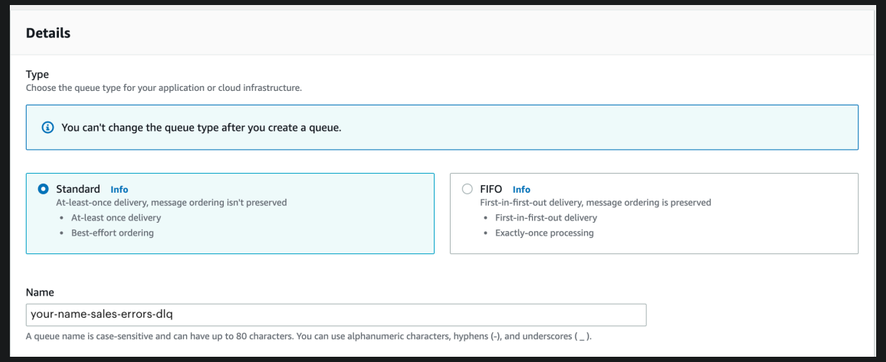
> How do we set up dead letter queues?

Steps on next slides.

School of Tech
part of accenture

Create Dead Letter Queue (DLQ)

- We follow the same method as we did to create a queue above
- Use a related name to before, but append with `-dlq`
  - e.g. `<your-name>-sales-errors-dlq`
- Ensure it is encrypted at rest as before
- Add a `Name` tag as before (with the `-dlq` name)
- Accept the other standard configuration and select `Create Queue` at the bottom of the screen



School of Tech
part of accenture

- Open your `<your-name>-coffee-sales-queue` and select the edit button at the top right of the page
- One of the optional features is 'Dead-letter-queue'.
  - Set it to `Enabled` and choose your `-dlq` queue created earlier in the dropdown list,
- Then click `Save`
- Your `<your-name>-sales-errors-dlq` will now receive messages which could not be consumed by your `<your-name>-coffee-sales-queue`

School of Tech
part of accenture

Features of queues

- Different queue solutions offer different features
- Use case may dictate technology choice

Features may include:

- Visibility timeout
- Message retention period
- Delivery delay
- Best-effort ordering
- First-in-first-out delivery
- Exactly-once processing
- At-least once delivery

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.

- High throughput
- Highly scalable as the traffic grows (few messages to 100,000 messages, same level of performance and latency)
- This elasticity comes with a cost, possible **duplication** and possibility of receiving **out of order**

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.

Pros and cons:

- Higher latency (need to buffer to make sure there is no duplicate)
- Lower throughput
- If you want multiple workers, you need to create message groups (subscription groups - and have to add a tag to your messages)

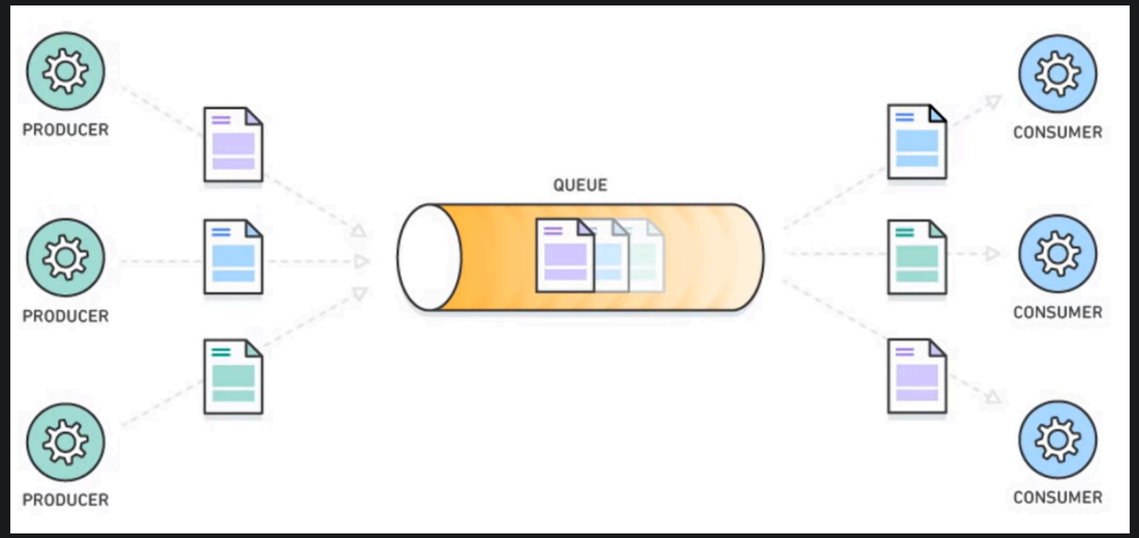
Emoji Check:

How did you find the concept of Queues, and it's implementation with AWS SQS?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

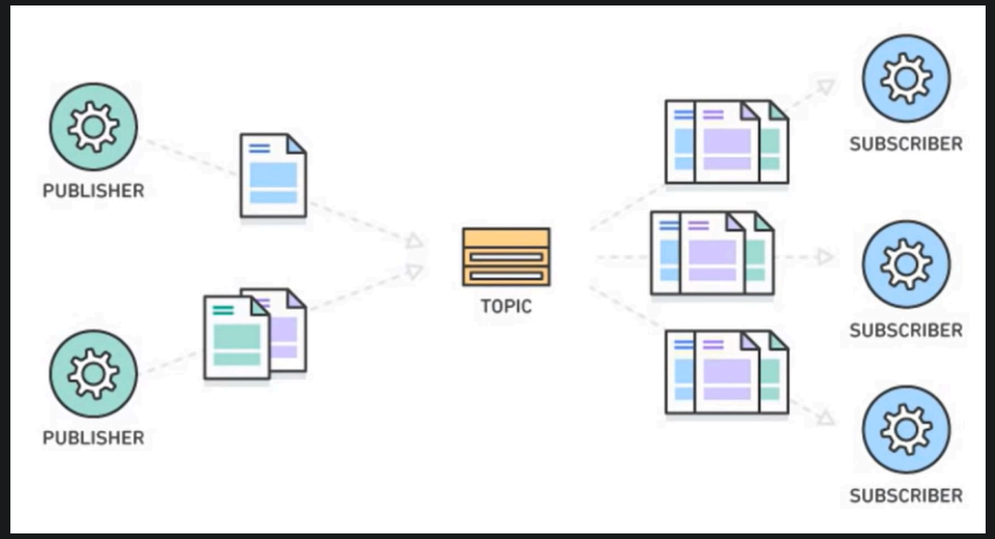
Queue vs Pub/Sub

Queue


PRODUCER
PRODUCER
PRODUCER
QUEUE
CONSUMER
CONSUMER
CONSUMER

School of Tech
part of accenture

Pub/Sub



Amazon Simple Notification Service (SNS)

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

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

SNS Topics

SNS introduces the concept of "topics" to solve these problems;

- Producers **Publish** to a topic
- Consumers **Subscribe** to that topic
- Hence the name **pub/sub**

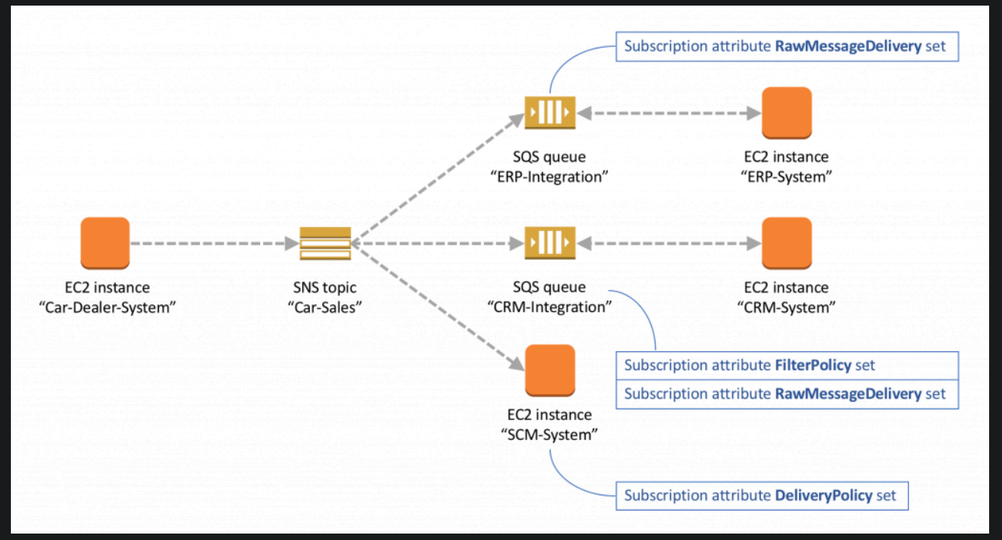
Multiple publishers

Notification queues can have multiple publishers.

> Can you think of examples where this might be useful?

Multiple Consumers

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



> Can you think of examples where this might be useful?

School of Tech
part of accenture

Subscriptions

Consumers can create subscriptions using a number of protocols:

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

School of Tech
part of accenture

SNS Demo - code along

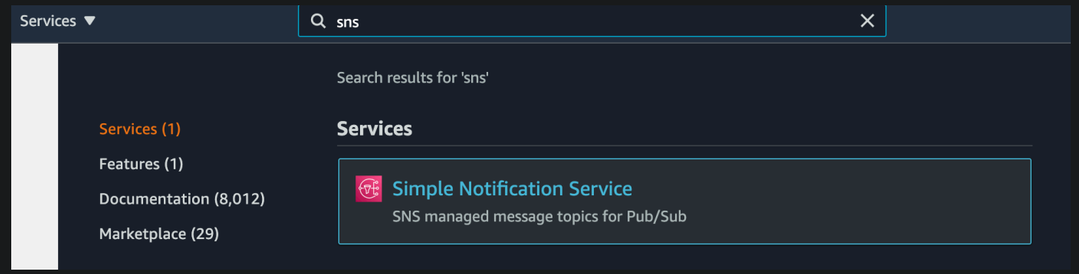
> A walkthrough setting up an SNS topic and pub/sub behaviour.

Steps on next slides.

School of Tech
part of accenture

Go to SNS in AWS

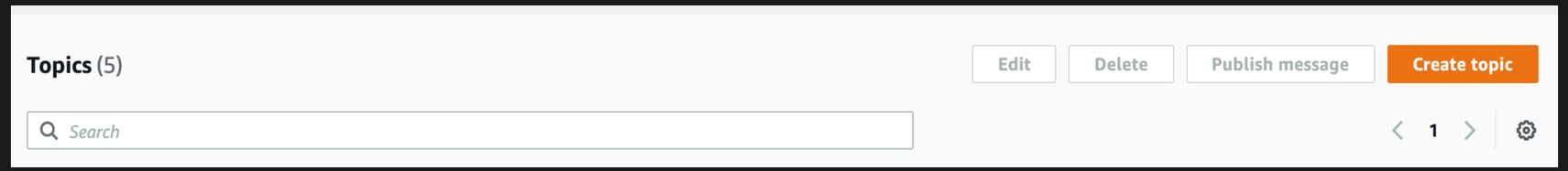
- Login to the AWS (Web) Console
- Click on "Services" and search for "SNS"
- ...and select "Simple Notification Service"



School of Tech
part of accenture

Create a Topic 1/2

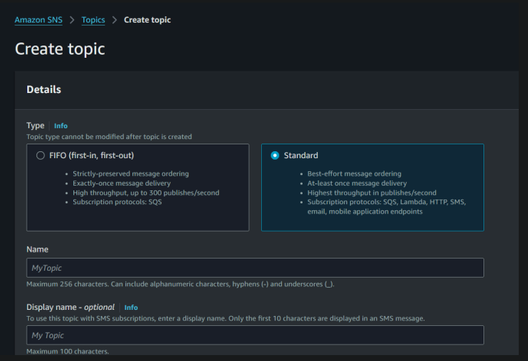
- Navigate to "Simple Notification System"
- Select "Topics"
- Select "Create Topic"



School of Tech
part of accenture

Create a Topic 2/2

- Choose Standard type
- Under `Encryption - optional` enable the `Encryption` option and keep the other defaults
- Enter a name e.g. `<your-name>-coffee-sales-notifications`
- Add a Tag with the key `Name` and same value as the queue name
- Click "Create"

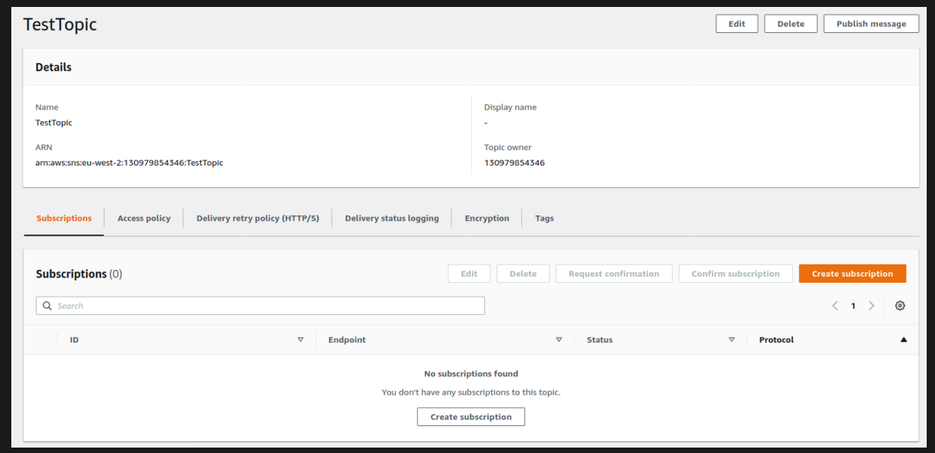


School of Tech
part of accenture

Create a Subscription

> Now we need to subscribe to our topic.

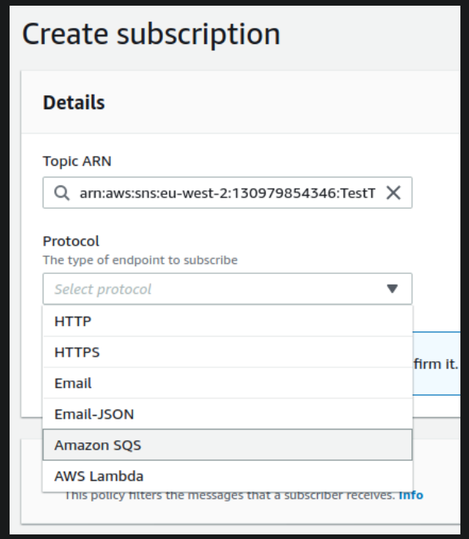
- You should now see the topic summary
- Click "Create Subscription"



School of Tech
part of accenture

Create a Subscription

- The topic name should be pre-populated
- Choose "Amazon SQS" as the protocol
- For "Endpoint" select `your-name-coffee-sales-queue` that you created earlier in SQS
- Click "Create Subscription"



School of Tech
part of accenture

Send a message to the topic

- Get the ARN of your topic
- We can submit a message using the AWS CLI using, for example,

```
aws sns publish --topic-arn <topic-arn> \
  --message "Latte for <your-name>"
```

You should get a response like:

```
{
    "MessageId": "54a2f084-c95d-56ab-9c42-9018e02b5f9d"
}
```

School of Tech
part of accenture

Errata: Debugging permissions

> You may need to add the `sqs:SendMessage` action to the access policy for your SQS queue. This should be set for you but does not always apply automatically.

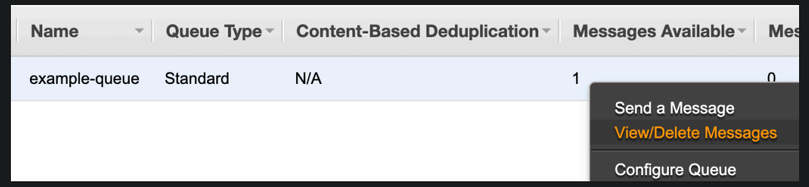
If you do need to add it:

- Find your Queue in SQS
- Click on the "Queue Policy" tab
- Edit the JSON of the Policy
- Add a Statement to the Policy like that found in ./handouts/errata-queue-policy.json
    - Substitute in the ARNS for your Queue and Topic
- Retry sending a Notification

School of Tech
part of accenture

Let's inspect the contents of the queue again

- 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



Success!

> Now we have sent a notification using the CLI, let is again try to use Python instead

Sending a message to topic with BOTO3

```
import boto3
import json

session = boto3.Session(profile_name='<aws-profile>')
sns_client = session.client('sns')

message = { "type": "sale", "item": "coffee", "price": 1.50 }
```

continued on next slide...

Sending a message to topic with BOTO3

```
response = sns_client.publish(
    TargetArn="<topic-arn>",
    Message=json.dumps({'default': json.dumps(message)}),
    MessageStructure='json'
)

print(f'response = {response}')
```

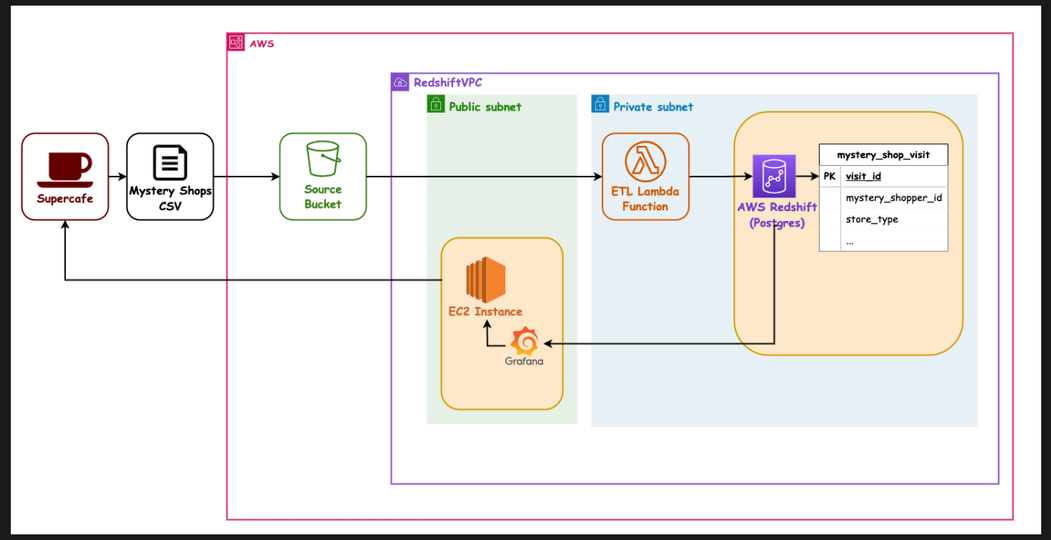
Going forward

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

School of Tech
part of accenture

Back to Mystery Shopper (again)

Consider the Mystery Shopper example;



Discussion - 5 mins

> **Where** in the Mystery Shopper architecture might we put notifications queues (SNS)?
>
> And - *why* would you put them there?
>
> What about the SQS?

School of Tech
part of accenture

Emoji Check:

How did you find the concept of Topics, and its implementation with AWS SNS?

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively

School of Tech
part of accenture

Things to be aware of

> There are a few extra things we need to consider in our systems is using queues...

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?

Message Throughput

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

Terms and Definitions - recap

Message: A broad term for data passed between services. Usually, but not always, as JSON.

Queue: A method of service-to-service communication where 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 it

School of Tech
part of accenture

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

Overview - recap

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

Learning Objectives - recap

- 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

Further Reading

- Intro to message queues
- Amazon SQS
- Intro to Event Driven Architecture
- Intro to the pub/sub design pattern
- Amazon SNS

School of Tech
part of accenture

Emoji Check:

On a high level, do you think you understand the main concepts of this session? Say so if not!

1. 😢 Haven't a clue, please help!
2. 😕 I'm starting to get it but need to go over some of it please
3. 😐 Ok. With a bit of help and practice, yes
4. 🙂 Yes, with team collaboration could try it
5. 😀 Yes, enough to start working on it collaboratively