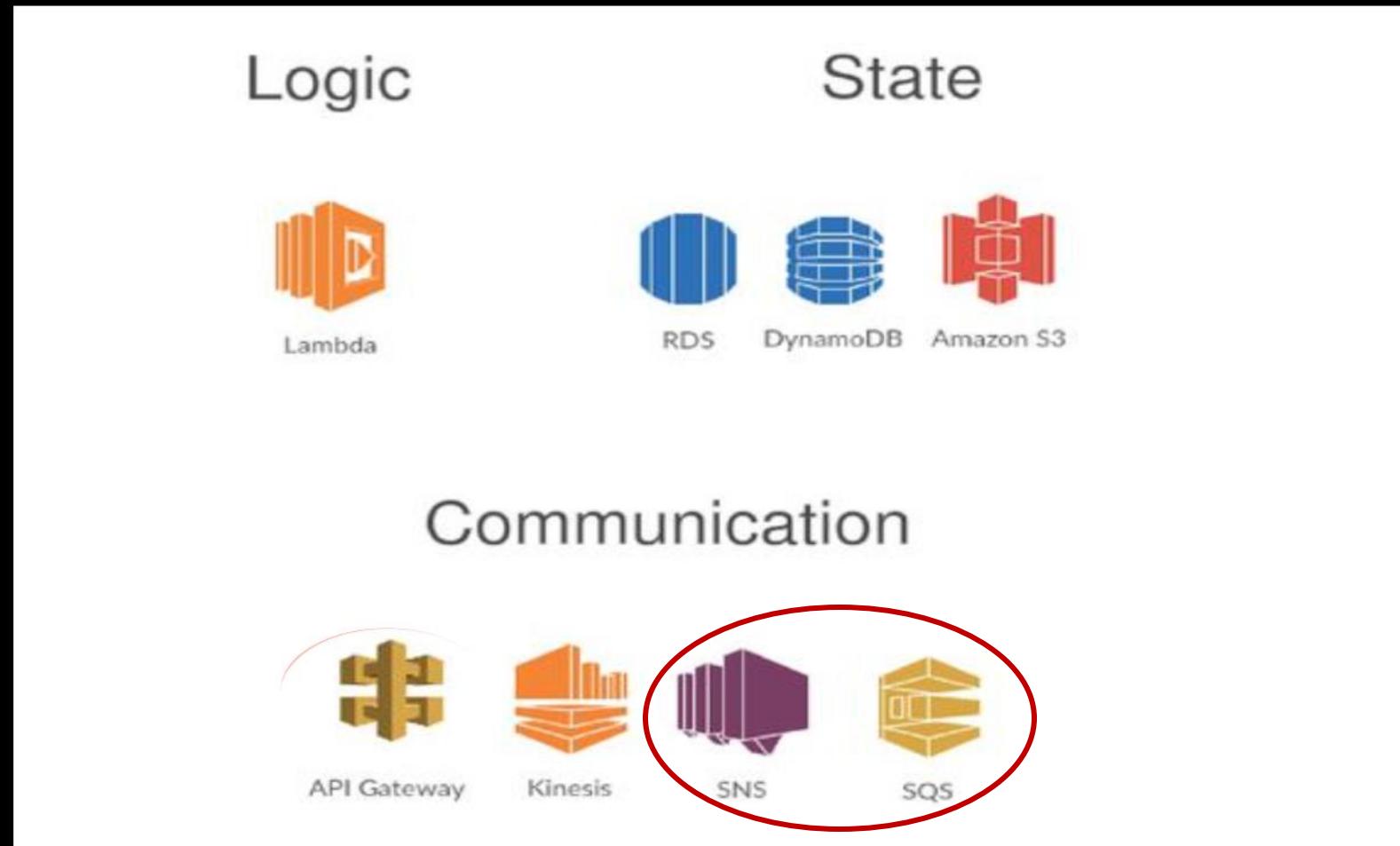
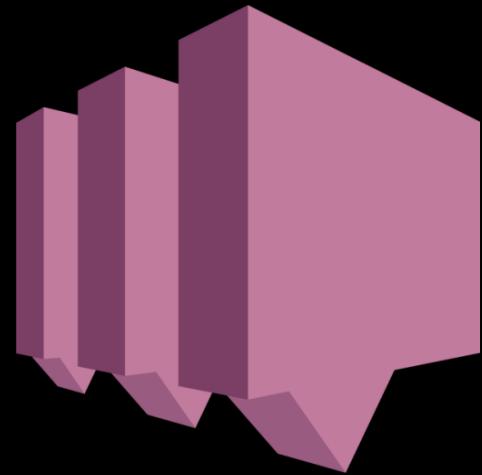


## AWS Integration and Messaging Services (Contd).

# Components of a Serverless, Message-Driven application (aka Event Driven Architecture - EDA)





# Simple Notification Service (SNS)

# Amazon SNS

- Released in 2010.
- A ‘serverless’ publish-subscribe (pub/sub) messaging service.
- When you want to send a message to many receivers.
  - SQS is point-to-point.
  - SNS is pub/sub.
- The publisher sends a message to an SNS topic.
- Many subscribers can listen to a topic.
- Each topic subscriber gets all the messages.
- Subscribers can be:
  - SQS, HTTP / HTTPS, Lambda function
  - Emails (SES)
  - SMS messages, Mobile Notifications

# SNS - Features

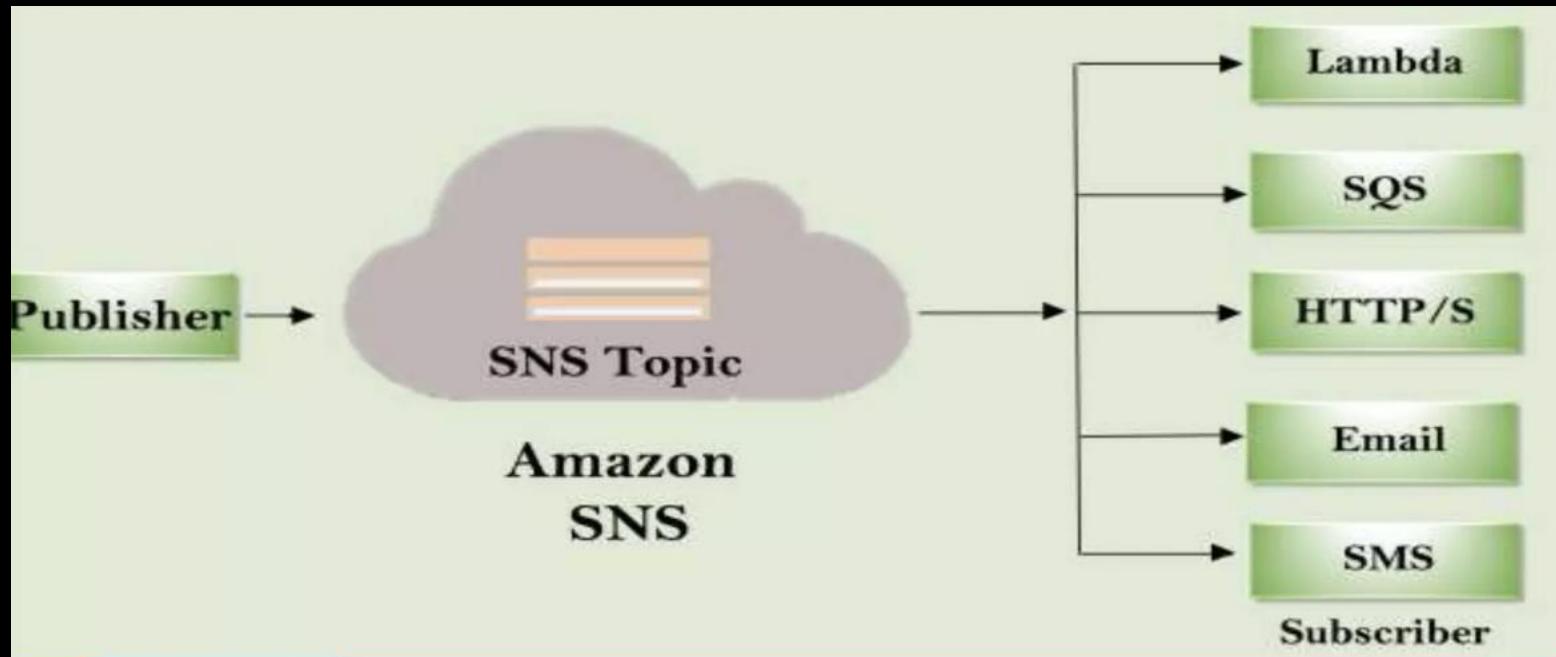
- Integrates with lots of services (Publishers):
  - Lambda.
  - S3 (Bucket change notifications).
  - Cloudwatch (Alarm notification).
  - etc
- Encryption:
  - In-flight encryption using HTTPS API.
  - At-rest encryption using KMS keys.
- Message Filtering:
  - Subscriber can declare a filtering policy to limit the messages it receives to those of interest.

# SNS - Features

- Security:
  - Access Controls: IAM policies to regulate access to the SNS API.
  - SNS Access Policies (similar to S3 bucket policies):
    - Cross-account access to SNS topics.
    - Allowing other services (e.g. S3) to write to an SNS topic.
- Auto-scaling.
- DLQ – an SQS queue for messages that can't be delivered to a subscriber due to client errors or server errors.

# Topics

- An SNS topic is a logical access point that acts as a communication channel.
- A topic lets you group multiple endpoints, e.g. SQS, Lambda, SMS



# Demo – CDK provisioning code

- Architecture:

AWS CLI (Publisher) → SNS Topic → Lambda (Subscriber)

```
23     const demoTopic = new sns.Topic(this, "DemoTopic", {
24         displayName: "Demo topic",
25     );
26
27     const processMessageFn = new lambdanode.NodejsFunction(
28         this,
29     >     "processMsgFn", ...
30     }
31     );
32
33     demoTopic.addSubscription(new subs.LambdaSubscription(processMessageFn));
34
35     new cdk.CfnOutput(this, "topicARN", {
36         value: demoTopic.topicArn,
37     );
38
39
40
41
42
43
```

# Demo – Lambda subscriber

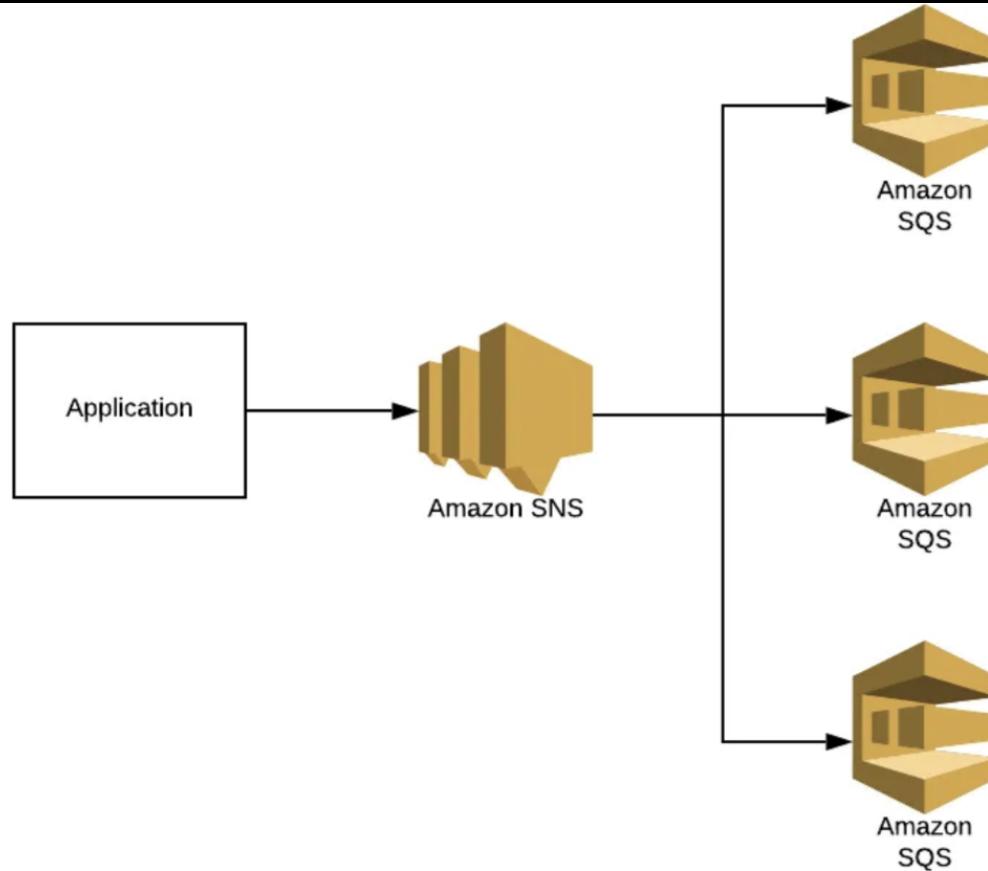
- A Lambda subscriber receives a batch (?) of messages in its event parameter.

```
aws sns publish \
--topic-arn "topic-arn" \
--message file://message.json
```

The screenshot shows the AWS Lambda function logs. A log entry is highlighted with a yellow background, showing a JSON message structure. The message contains a timestamp, a unique identifier, and a 'Records' array. One record is shown in detail, containing an 'EventSource' (aws:sns), 'EventVersion' (1.0), 'EventSubscriptionArn' (arn:aws:sns:eu-west-1:517039770760:SimpleAppStack-DemoTopic2BE41B12-H01c5byPIZhM:7c95678d-1044-4b6f-9b7f-9a09a59f8b4e), and an 'Sns' object. The 'Sns' object includes a 'Type' (Notification), 'MessageId' (28fab4c1-7978-5134-a043-e91e95ab2d8f), 'TopicArn' (arn:aws:sns:eu-west-1:517039770760:SimpleAppStack-DemoTopic2BE41B12-H01c5byPIZhM), 'Subject' (null), and a 'Message' field. The 'Message' field is a JSON object with 'name' (Diarmuid O' Connor), 'address' (1 Main Street), 'email' (doconnor@wit.ie), and a timestamp (2023-11-28T13:26:43.682Z). The message also includes a signature and a signing certificate URL. A red box highlights the 'Message' field.

```
2023-11-28T13:26:43.682Z      06899e30-1c7d-4a1
{
  "Records": [
    {
      "EventSource": "aws:sns",
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:eu-west-1:517039770760:SimpleAppStack-DemoTopic2BE41B12-H01c5byPIZhM:7c95678d-1044-4b6f-9b7f-9a09a59f8b4e",
      "Sns": {
        "Type": "Notification",
        "MessageId": "28fab4c1-7978-5134-a043-e91e95ab2d8f",
        "TopicArn": "arn:aws:sns:eu-west-1:517039770760:SimpleAppStack-DemoTopic2BE41B12-H01c5byPIZhM",
        "Subject": null,
        "Message": "{\n          \"name\" : \"Diarmuid O' Connor\",\n          \"address\" : \"1 Main Street\",\n          \"email\" : \"doconnor@wit.ie\"\n        }",
        "Timestamp": "2023-11-28T13:26:43.682Z",
        "SignatureVersion": "1",
        "Signature": "KdEyxqPvp0d6TD59FCojNYat4+KleQdZIomAs7ULcsxw9GUMoei4ftUHFLu2IfIn8KWZWSMNr2g8M3ZfLeadoTNQCbe2kWhA5aS4r3Cvj68WJkusvCUpoVyrnzPJMN5HNn+D2GL4VVvf7IN1VvfH34Y14i7jUHHWTbgEQouD7lTF1CjkLR09bCNoe0JDFprp1nQQJ0LAqtDm+52+d+29+pZ0f61he1xo2i6rSLxj4VZ30mFyrPwKBwgHCdSQfQQ4/x4U1mZZdG/sbXcIdy5yznKBmrjmnnvHFLyffFz5xqiuBnGHhymzyiGSVOhmBBFNMcigABTUVEpAvI01/PYEDw==",
        "SigningCertUrl": "https://sns.eu-west-1.amazonaws.com/SimpleNotificationService-01d088a6f77103d0fe307c0069e40ed6.pem"
      }
    }
  ]
}
```

# The Fan-out pattern



Each Queue  
has a different  
Lambda function  
consumer

# Demo – Fan Out.

- The Fan Out subscribers can be a mixture of types.
- Demo Architecture:

AWS CLI (Pub) -→ SNS Topic -→ Lambda (Sub)  
-→ SQS (Sub). → Lambda (Consumer)

```
const demoTopic = new sns.Topic(this, "DemoTopic", {});  
const queue = new sqs.Queue(this, "all-msg-queue", {});  
const processSNSMessageFn = new lambdanode.NodejsFunction(  
    this,  
    "processSNSMsgFn",  
    {  
        ... properties .....  
    }  
);  
// Subscribers  
demoTopic.addSubscription(new subs.LambdaSubscription(processSNSMessageFn,  
    { ... properties ..... }));  
  
demoTopic.addSubscription(new subs.SqsSubscription(queue,  
    { ... properties ..... }));
```

# Demo – The Lambda subscriber event parameter

The screenshot shows a Chrome browser window with multiple tabs open, including CloudWatch, docker-compose, Mail - Diarmuid, Module: Agile S, Grammarly, and Event\_Driven\_. The main content is the AWS CloudWatch Logs console for the eu-west-1 region.

The left sidebar shows the CloudWatch service navigation with options like Alarms, Logs (selected), Log groups, Log Anomalies, Live Tail, Logs Insights, Contributor Insights, Metrics, X-Ray traces, Events, and Application Insights.

The logs list shows a single log entry from 2024-11-13T14:11:16.529Z. The log content is a JSON object:

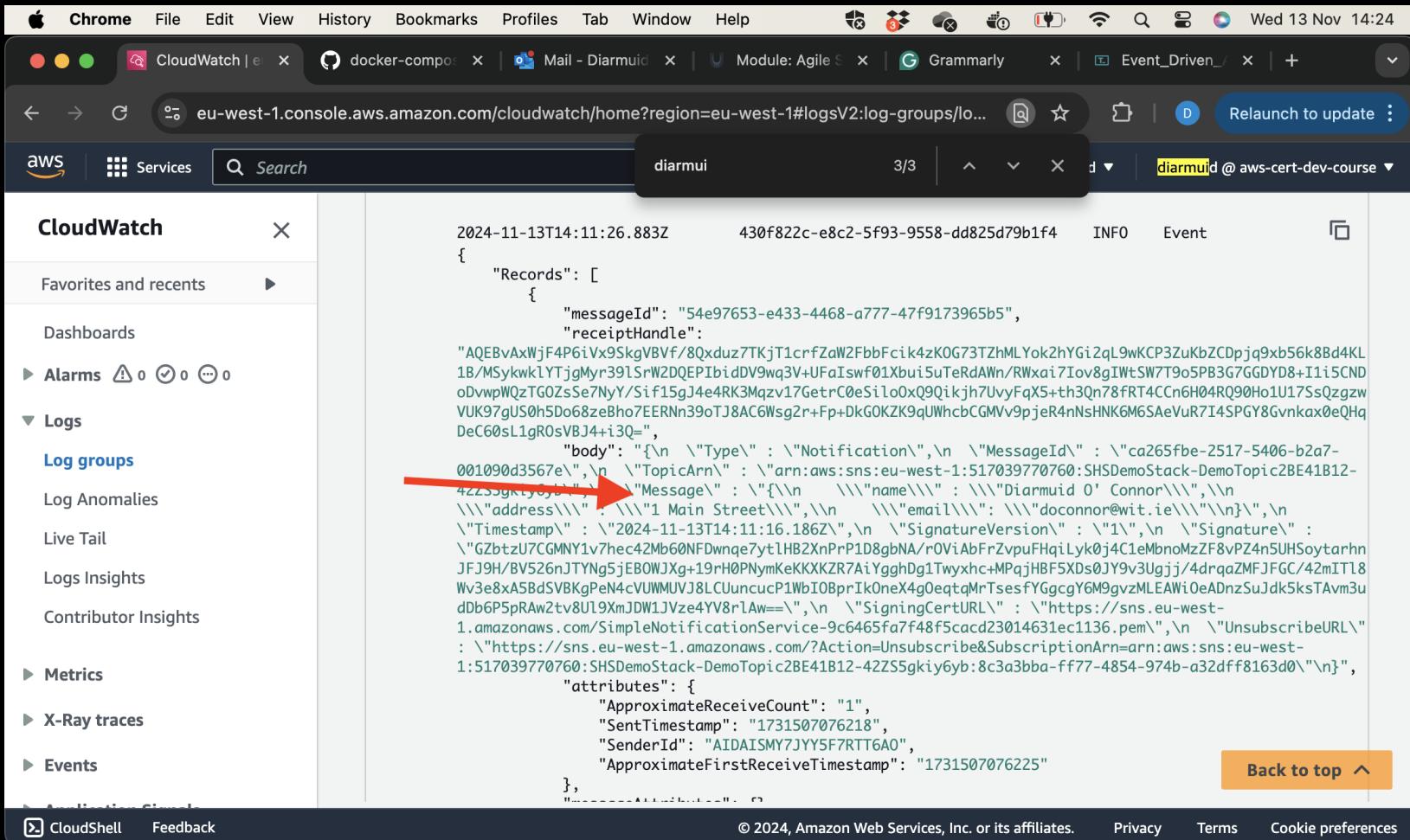
```
1  {
2      "name" : "Diarmuid O' Connor",
3      "address" : "1 Main Street",
4      "email": "doconnor@wit.ie"
5  }
```

A red arrow points to the "Records" field in the JSON object, which contains the actual event data. The event data includes details about the sns message, such as the subject being null and the message body containing the subscriber's information.

At the bottom right of the page, there is a "Back to top" button.

Page footer: © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

# Demo – The Lambda Q consumer event parameter



The screenshot shows the AWS CloudWatch Logs interface in a Chrome browser. The left sidebar is titled "CloudWatch" and includes sections for "Logs" (selected), "Metrics", "X-Ray traces", and "Events". The main pane displays a log entry with the following details:

- Timestamp: 2024-11-13T14:11:26.883Z
- Log ID: 430f822c-e8c2-5f93-9558-dd825d79b1f4
- Level: INFO
- Type: Event

The log body is a JSON object:

```
{  
  "Records": [  
    {  
      "messageId": "54e97653-e433-4468-a777-47f9173965b5",  
      "receiptHandle": "AQEBvAxWjF4P6iVx9SkqVBVF/80xduz7TkjT1crfZaW2FbbFcik4zK0G73TzhMLYok2hYGi2ql9wKCP3ZuKbZCDpj9xb56k8Bd4KL  
1B/MSykwkLYTjgMyr391SrW2DQEPIbidDV9wq3V+UFaIswf01Xbu15uTeRdAWn/RWxai7Iov8gIWtSW7T9o5PB3G7GGDYD8+I1i5CND  
oDwpwWQzTG0ZsSe7NyY/Sif15gJ4e4RK3Mqzv17GetrC0eSilo0xQ9Qikjh7UvyFqX5+th3Qn78FRT4CCn6H04RQ90Ho1u17SsQzgzw  
VUK97gJ50h5D68zeBho7EERNh39oTJ8AC6Wsg2r+Fp+DkGOKZK9qUWhcbCGMV9jeR4nNsHNK6M6SAeVuR7I4SPGY8Gvnkax0eQh  
DeC60sL1gR0sVBj4+i3Q=",  
      "body": "\n        \"Type\": \"Notification\",  
        \"MessageId\": \"ca265fbe-2517-5406-b2a7-  
001090d3567e\",  
        \"TopicArn\": \"arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-  
42ZS5gk1yjyj\",  
        \"Message\": \"\\n        \"name\": \"Diarmuid O' Connor\",\\n  
        \"address\": \"1 Main Street\",\\n        \"email\": \"doconnor@wit.ie\",\\n  
        \"Timestamp\": \"2024-11-13T14:11:16.186Z\",\\n        \"SignatureVersion\": \"1\",\\n        \"Signature\": \"GbtzU7CGMNY1v7hec42Mb60NFdwne7ylHB2XnPrP1D8gbnA/0ViAbFrZvpfHqjLyk0j4C1eBn0mZF8vPZ4n5UHSoytarhn  
JFJ9H/BV526nJTYNg5jEBOWJXg+19rH0PNymKeKKXKR7AiYgghdg1Twyxhc+MPqjHBF5XD0s0Y9v3Ugjj/4drqaZMFJFGC/42mITl8  
Wv3e8xAS8dSVBKgPeN4cVUWMUVJ8LCUuncucP1WbIOBpriK0neX4g0eqtqMrTseseYGgcgY6M9gvzMLEAWi0eADnzSuJdk5ksTAvm3u  
dBb6P5pRAw2tv8Ul9XmJDW1JVze4YV8rlAw==\",\\n        \"SigningCertURL\": \"https://sns.eu-west-  
1.amazonaws.com/SimpleNotificationService-9c6465fa7f48f5cad23014631ec1136.pem\",\\n        \"UnsubscribeURL\": \"https://sns.eu-west-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:eu-west-  
1:517039770760:SHSDemoStack-DemoTopic2BE41B12-42ZS5gk1yjy6yb:8c3a3bba-ff77-4854-974b-a32dff8163d0\"\\n    \"\",  
        \"attributes\": {  
          \"ApproximateReceiveCount\": \"1\",  
          \"SentTimestamp\": \"1731507076218\",  
          \"SenderId\": \"AIDAISMY7JYY5F7RTT6AO\",  
          \"ApproximateFirstReceiveTimestamp\": \"1731507076225\"  
        },  
        \"\"  
    }  
  ]  
}
```

A red arrow points to the "body" field of the log entry, highlighting the JSON payload.

# The SNS envelope.

- SNS wraps the source message in an envelope before sending it to an SQS queue subscriber.
  - Configurable

```
demoTopic.addSubscription(new subs.SqsSubscription(queue, {  
    rawMessageDelivery: true,  
});
```

# The SNS envelope.

```
2023-11-29T12:49:03.995Z          f0e106dd-9997-510c-9148-5b3caccf8f57    INFO    Event
{
  "Records": [
    {
      "messageId": "192a3ef6-176d-4ce9-a9fa-7f9994582a88",
      "receiptHandle": "AQEB8goKUG1tRGemA3nrBYMBPQyFHTDH2JJA5u6hd+mwZ+RxsNu3IlszA9uKurF+uY3mHr8XnofiMJS2Zx1j
iy2nS6ohVV016mh1Cwq64dla3JX1L+RIpcqxNhOF0qMzK56kF36B1zSIxZ500XaviIiHx0xtr0Fswp+u1n7hJ
+0TxBkW/V81c/b+jRdM611Hn7hqKb5V2xXkv/AJgfEo2sWdz5SVS8BLMDyIEGMmEngKq4Tnbpe0jiJpydUCX0
Z1zrEoaWoqrWxD0kX1P7fqQxZ73zGYkMHCvD/01bUIkKnywzgvTHDF2Fj2bCDNLsMRrq4qFDGoIAwx4VOP14
8ZP8DEpiYlnTSEEEmA2AAK8oyDJ4AGNdp+lNidAWZiOGFau6Uj1uGw81h7iuFWOA1iOnRd1rndK0x6NosE/5xsY
leSyWZM=",
      "body": "{\n  \"Type\" : \"Notification\", \"MessageId\" : \"2625cdf8-
6f50-5eb3-aaaa-90195a3bce66\", \"TopicArn\" : \"arn:aws:sns:eu-west-
1:517039770760:SHSDemoStack-DemoTopic2BE41B12-paPr90UK7P0D\", \"Message\" : \"\n    \":name\": \"Diarmuid O' Connor\", \":address\": \"1
Main Street\", \":email\": \"doconnor@wit.ie\"\n  \"Timestamp\" : \"2023-11-29T12:48:43.515Z\", \"SignatureVersion\" : \"1\", \"Signature\" :
\"cW3s3K1SJq1HBqhiaNrC3QEbxJZBR/g1b0C0QFk5eRkPKp2j8gGYkEGISi0eerdgd+Pff9lo1M1NuGiYI7
0q3k9b0Fw9jkIh41+5tMnskDQk9mr/mdLHYfjIK2wmenMa7hggScsgWNfQtplnt4Z8EGWrwA9lrlNIRtLFHTkS
YuY/m9FdGwe3dDC8AYmsui8WzFP74vyPv46JkIgKDunqy4YsqUXdbCA2Hv7j/LV1WqXMKX21+6Hi8DF+u3q9l
1YzPWboTgbVlgWvzqbPFuY6tb6Z6yLEZi/ud00YitgiaoilWl8X9SEGqpnuo25+mIGgjM6AjVUGgbgWbiYU4wl
Mhv==\", \"SigningCertURL\" : \"https://sns.eu-west-
1.amazonaws.com/SimpleNotificationService-01d088a6f77103d0fe307c0069e40ed6.pem\", \"UnsubscribeURL\" : \"https://sns.eu-west-1.amazonaws.com/?
Action=Unsubscribe&SubscriptionArn=arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-
DemoTopic2BE41B12-paPr90UK7P0D:dd9b7420-ef0e-4e1b-88a2-e896b98d6612\"\n  }",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1701262123540",
        "SenderId": "ATNATCMV71VVVF7DTT6AO"
      }
    }
  ]
}
```

**SNS envelope**

**Copy**

**Back to top ▾**

# The SNS envelope.

	▶	2023-11-29T12:54:51.889+00:00	INIT_START Runtime Version: nodejs:16.v26 Runtime Version ARN...
	▶	2023-11-29T12:54:52.043+00:00	START RequestId: b1143799-4476-5d78-a682-6a7872b6f2c7 Version...
	▼	2023-11-29T12:54:52.045+00:00	2023-11-29T12:54:52.045Z b1143799-4476-5d78-a682-6a7872b6f2c7...
		2023-11-29T12:54:52.045Z	b1143799-4476-5d78-a682-6a7872b6f2c7 INFO Event
		{	<button>Copy</button>
		"Records": [	
		{	
		"messageId": "973db4ca-53e2-4f29-ad3c-7bc252b02e25",	
		"receiptHandle":	
		"AQEbx/TenyP7WL3SsdK17/QBif3japCb6NjIG0iLt+hDIXEvW0ps+ZP05V7PFFx+Cgq/0lwUQJW6xWCJfgZ4	
		9feLyz5cxKBEwGL27kH1IY7roBcxgGDgbK/TbIcAbzVEcpoeCxmTdWCYzzvE66grzz7jEVEDbnEkV6l0y+gXV	
		No SNS envelope xeMtydar80E1989Hm5qBCrn7oG4T4FPamGZh907GOUJnVZPK8cSTtTNTATk8/HrcW	
		JYevV9Mp3tvd00L0qk3rm0ZZdz0PYRX0Ew7Uj9/D40jleOR+asKc0lsNetoJfqB1	
		FEHOIk1yvp4WPe0k9L0HGxkN0TImzg8yJzRrJL60ge3K3CRwopapt5w6giWYlRW+KTGm11s5+HgrDXrQFWl03	
		fDZoYPo=",	
		"body": "{\n                \"name\": \"Diarmuid O' Connor\",\n                \"address\" : \n            \"1 Main Street\",\n                \"email\": \"doconnor@wit.ie\"\n        },	
		"attributes": {	
		"ApproximateReceiveCount": "1",	
		"SentTimestamp": "1701262471745",	
		"SenderId": "AIDAISMY7JYY5F7RTT6A0",	
		"ApproximateFirstReceiveTimestamp": "1701262471751"	
		},	
		"messageAttributes": {},	
		"md5OfMessageAttributes": null,	
		"md5OfBody": "85f8fd703039e25159f4268695f0cd5f",	
		"receiptHandle": "	
			<a href="#">Back to top ^</a>

# Lambda Vs { SQS → Lambda } subscribers

- Disadvantages (Lambda subscriber)
  - No batching is available when processing messages from SNS.
  - No control of Lambda Concurrency,
    - i.e. messages are processed one by one as soon as they arrive.
  - Lambda function is responsible for handling errors/retries
    - A DLQ needs to be configured.
- Advantages (Lambda subscriber):
  - Good for time-critical processing.

# SNS - Delivery protocols and policies.

- SNS defines a delivery policy for each delivery protocol (subscriber type).
- The policy defines how SNS retries the delivery of messages when server-side errors occur,
  - i.e. when the service that hosts the subscriber is unavailable/not responding.
- . When the delivery policy is exhausted, SNS stops retrying and discards the message.
  - A DLQ can be assigned for this case.

# SNS - Delivery protocols and policies.

Endpoint type	Delivery protocols	Immediate retry (no delay) phase	Pre-backoff phase	Backoff phase	Post-backoff phase	Total attempts
AWS managed endpoints	Amazon Kinesis Data Firehose <sup>1</sup>	3 times, without delay	2 times, 1 second apart	10 times, with exponential backoff, from 1 second to 20 seconds	100,000 times, 20 seconds apart	100,015 times, over 23 days
	AWS Lambda					
	Amazon SQS					
Customer managed endpoints	SMTP	0 times, without delay	2 times, 10 seconds apart	10 times, with exponential backoff, from 10 seconds to 600 seconds (10 minutes)	38 times, 600 seconds (10 minutes) apart	50 attempts, over 6 hours
	SMS					
	Mobile push					

# Lambda subscriber DLQ

- SNS invokes a lambda function subscriber asynchronously.
  - SNS does not wait for a response.  
⇒ Lambda service must handle function failures cases.

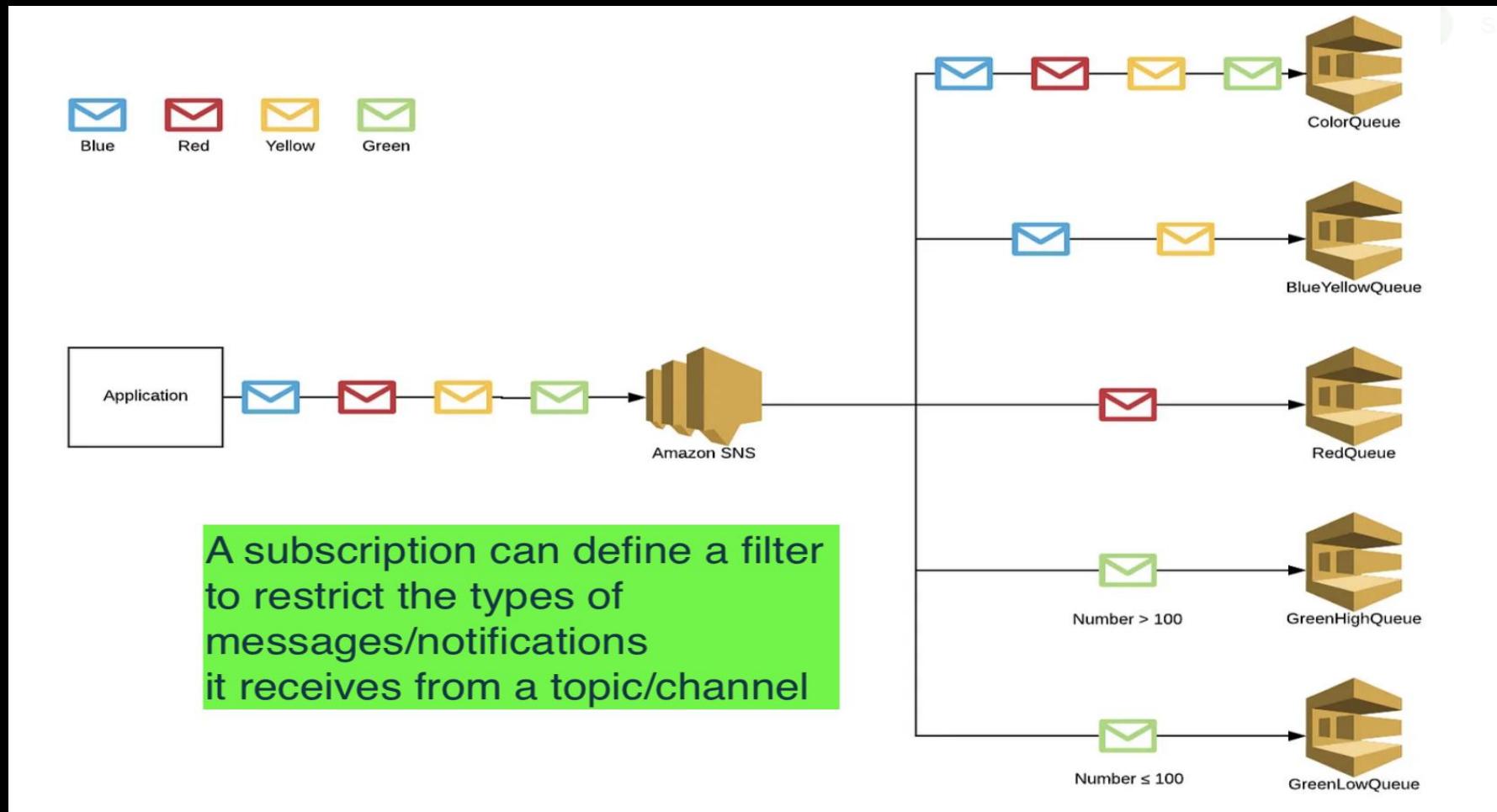
- Ex.: Architecture:

AWS CLI (Pub) → SNS Topic → Lambda (Subscriber)

|

| → DLQ → Lambda (Consumer)

# Fan Out pattern - Filtering.



# Filtering

- The Filtering policy can be based on:
  1. Attributes of the message or
  2. The message body.
- Filtering criteria options:
  1. String Filter.
    - Conditions - allowList, denyList, matchPrefixes
  2. Numeric Filter.
    - Conditions – allowList, greaterThan, lessThan, between.
  3. Exists Filter.

→ DLQ → Lambda (Consumer)

# Demo – Message Attribute Filtering

- Architecture:  
AWS CLI (Pub) → SNS Topic → <Filter> → Lambda (Sub)  
→ <Filter> → SQS Q -→ Lambda (Con)
- Filter policy - The Lambda subscriber should only receive messages with a user\_type attribute set to Student or Lecturer.

# Demo – Message Attribute Filtering

```
demoTopic.addSubscription(  
    new sns.SubscriptionBuilder(processSNSMessageFn, {  
        filterPolicy: {  
            user_type: sns.SubscriptionFilter.stringFilter({  
                allowlist: ["Student", "Lecturer"],  
            }),  
        },  
    })  
);
```

```
{} attributes.json > ...  
1  {  
2      "user_type": {  
3          "DataType": "String",  
4          "StringValue": "Lecturer"  
5      },  
6      "source": {  
7          "DataType": "String",  
8          "StringValue": "Moodle2023"  
9      }  
10 }
```

```
aws sns publish --topic-arn topic-arn-value \  
    --message-attributes file://attributes.json \  
    --message file://message.json
```

```
{} message.json > ...  
1  {  
2      "name" : "Diarmuid O' Connor",  
3      "address" : "1 Main Street",  
4      "email": "doconnor@wit.ie"  
5  }
```

# Demo – Message Attribute Filtering

```
2024-11-19T13:44:32.907Z      5e23c426-17a8-4626-9da0-a2e8a95bfdbe      INFO      Event      □
{
  "Records": [
    {
      "EventSource": "aws:sns",
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-
MKUNYvCT2hVh:ab401e88-a6da-4c76-8b00-37e74c1b8f30",
      "Sns": {
        "Type": "Notification",
        "MessageId": "b96ba8e2-9ded-5563-801b-f754056d9c36",
        "TopicArn": "arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-
MKUNYvCT2hVh",
        "Subject": null,
        "Message": "{\n          \"name\" : \"Diarmuid O' Connor\",\n          \"address\" : \"1 Main\nStreet\",\\n\n          \"email\" : \"doconnor@wit.ie\"\n        }",
        "Timestamp": "2024-11-19T13:44:32.548Z",
        "SignatureVersion": "1",
        "Signature": "c0b04CTqDRbpBBnIi6+SMv1CAD6pLxuR/oWrLOEUjR5sUg80Llk66M8G+o2Qr5N32Dv2o6jdXLhiq27r5KbQCD6YknrDjXCm2CHUZ5
FI1E4aYPqWgULehNAIRTh0gp0Az1kLbAJZhpKmfpartMaAZn0iW3CqolPXTJjiEp1yxEsAuo4yldsmEDvrkzHE+A9r/ZpLW23W5kLW
PhLB8+0XuoI2bYC3prUiM6UXHxi9/hF4EI7HwLoL8IPTu7//6KtAkrIWUe41P1nDvtxDfUNKxVRn8HPIIz/woUj09rJeXdnRrhVgYdK
Roji53UqjBCm4drhI0FF968spmpa25LFwQ==",
        "SigningCertUrl": "https://sns.eu-west-1.amazonaws.com/SimpleNotificationService-
9c6465fa7f18f5cacd23014631ec1136.pem",
        "UnsubscribeUrl": "https://sns.eu-west-1.amazonaws.com/?Action=Unsubcribe&SubscriptionArn=arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-
MKUNYvCT2hVh:ab401e88-a6da-4c76-8b00-37e74c1b8f30",
        "MessageAttributes": {
          "user_type": {
            "Type": "String",
            "Value": "Lecturer"
          },
          "source": {
            "Type": "String",
            "Value": "Moodle2023"
          }
        }
      }
    }
  ]
}
```



To Be Continued