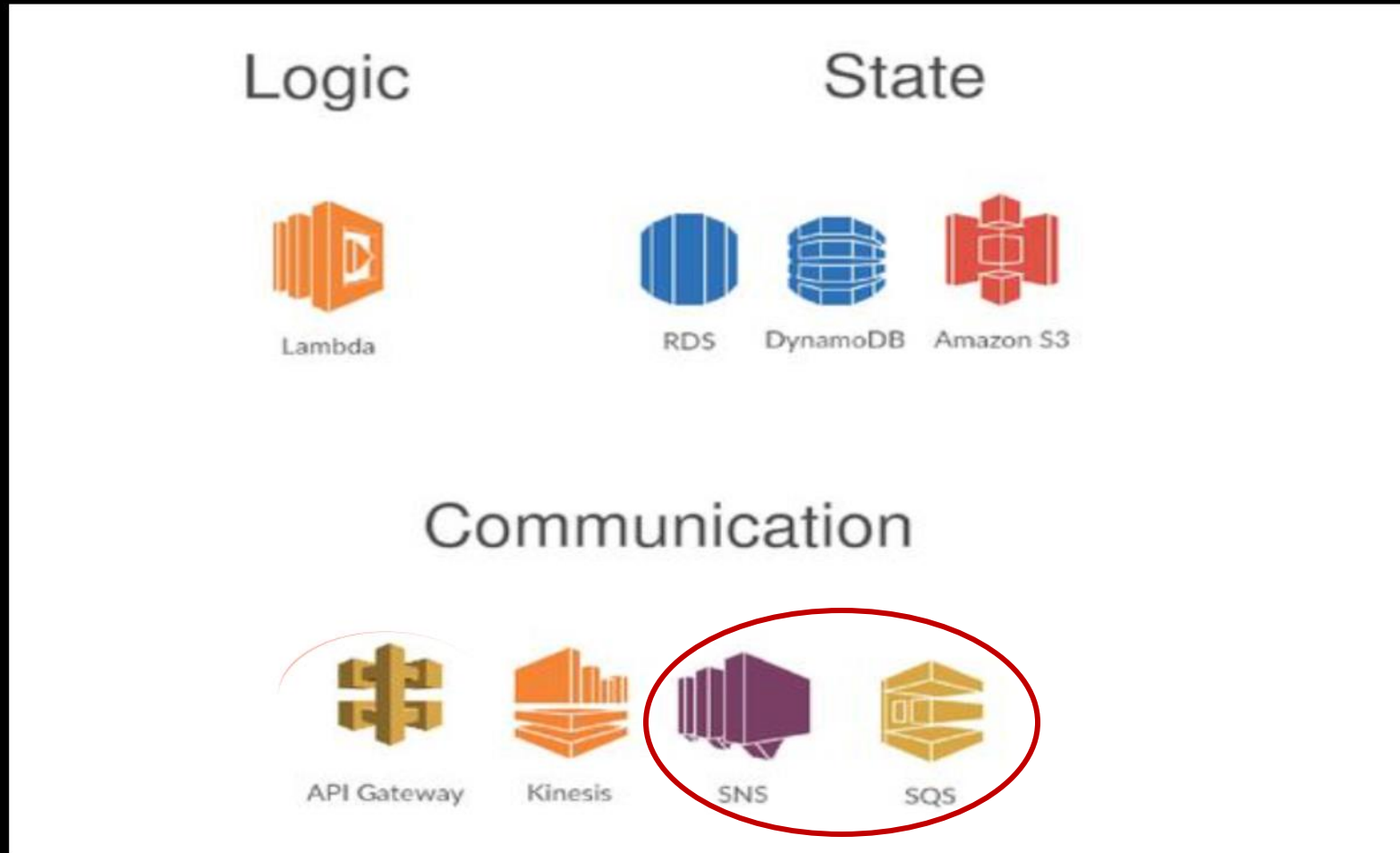
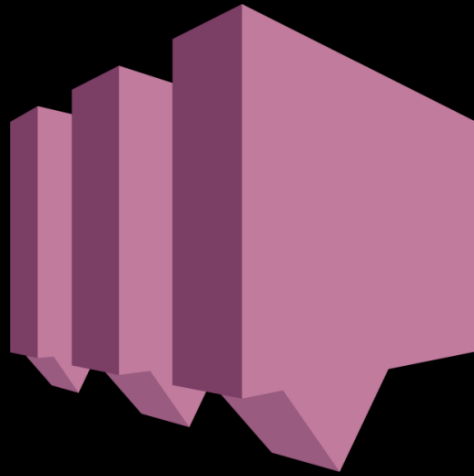


AWS Integration and Messaging Services (Contd).

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, but SNS is pub/sub.
- The publisher (event producer) sends a message to an SNS topic.
- Many subscribers can listen to the topic.
- Each topic subscriber gets all the messages.
- Subscribers can be:
 - SQS, HTTP / HTTPS, Lambda function
 - Emails (SES)
 - SMS messages, Mobile Notifications

SNS - Features

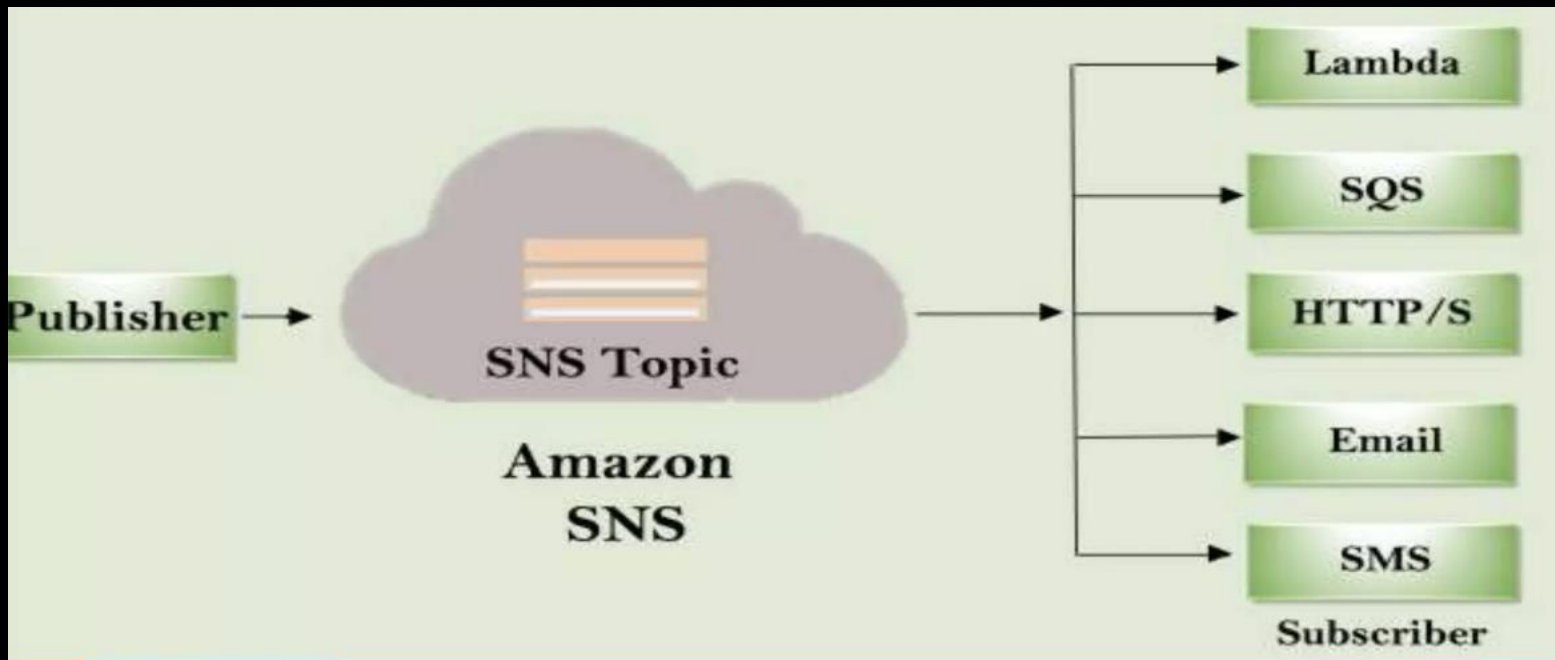
- Integrates with lots of services (Publishers):
 - Lambda.
 - S3 (Bucket change notifications).
 - Cloudwatch (Alarm notification).
 - etc
- Encryption:
 - In-flight encryption using HTTPS API.
 - At-rest encryption using KMS keys.
- Message Filtering:
 - Subscriber can declare a filtering policy to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



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", ...
35         }
36     );
37
38     demoTopic.addSubscription(new subs.LambdaSubscription(processMessageFn));
39
40     new cdk.CfnOutput(this, "topicARN", {
41         value: demoTopic.topicArn,
42     });
43
```

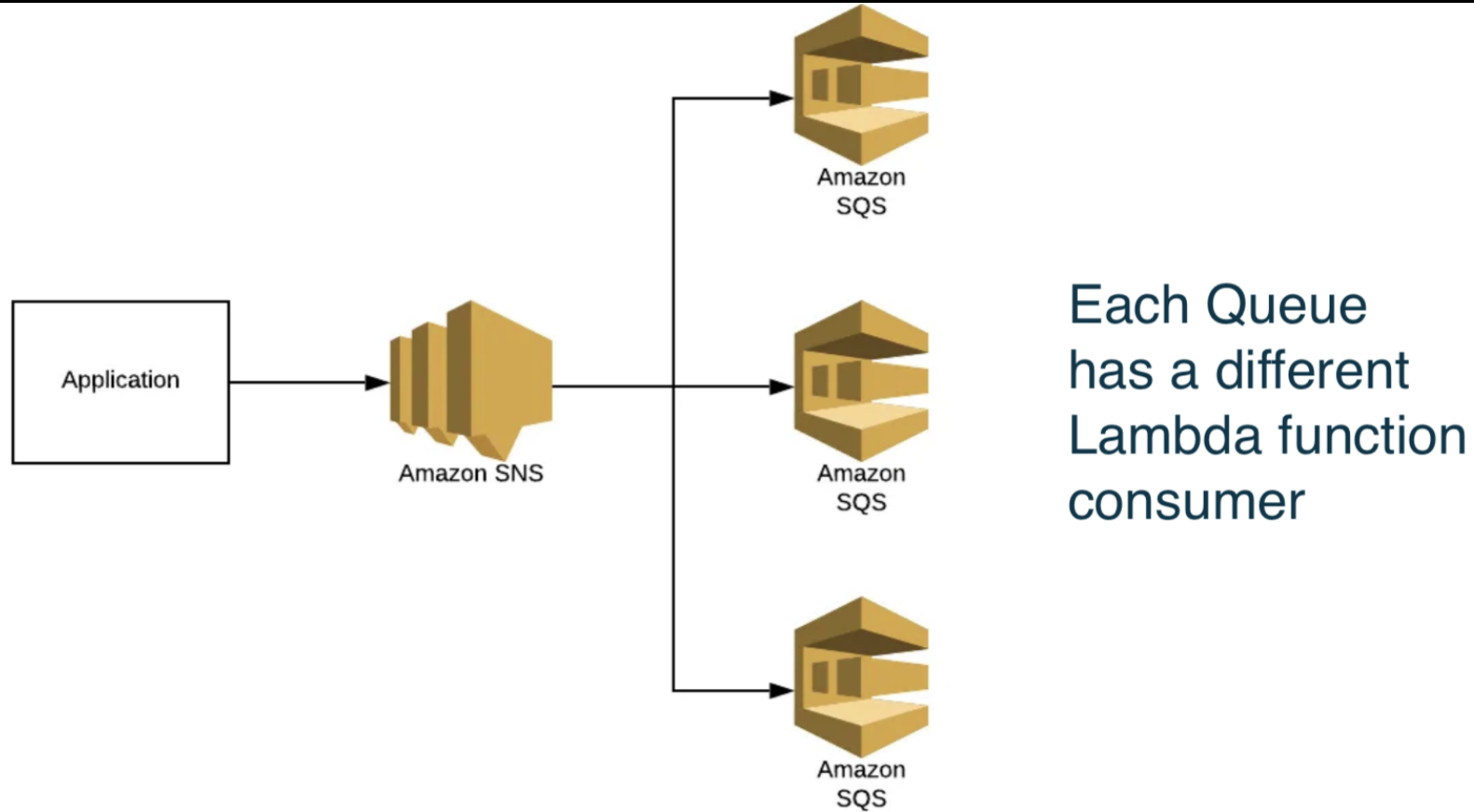

Demo – Lambda subscriber

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

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

```
2023-11-28T13:26:43.682Z      06899e30-1c7d-4a17-bdeb-  
{  
  "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\",  
  \"address\": \"1 Main Street\",  
  \"email\": \"doconnor@wit.ie\"  
}",  
        "Timestamp": "2023-11-28T13:26:43.682Z",  
        "SignatureVersion": "1",  
        "Signature":  
        "KdEyxpVvp0d6TD59FCojNYat4+KleQdZIomAs7ULcsxw9GUMoei4ftUHfLu2IfIn8KWZWSMnr2g8M3ZfLead  
oTNQCbe2kWhA5aS4r3Cvj68WJkusvCUppoVyrnzPJMNSHNn+D2GL4VVvf7IN1VvfH34Y14i7jUHHWTbgEQouD  
7lTF1CjkLR09bCNoe0JDFprp1nQQJ0LAqtDm+52+d+29+pZ0f61he1xo2i6rSLxj4VZ30mFyrPwKBwgHCdSQf  
QQ4/x4U1mZZdG/sbXcIdy5yznKBmrjmnivHfLyfFz5xqiuBnGHhymzyiGSV0hmBBFNMciGABTUVepAvI01/PY  
EDW==",  
        "SigningCertUrl": "https://sns.eu-west-  
1.amazonaws.com/SimpleNotificationService-01d088a6f77103d0fe307c0069e40ed6.pem",
```

The Fan-out pattern



Demo – Fan Out.

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

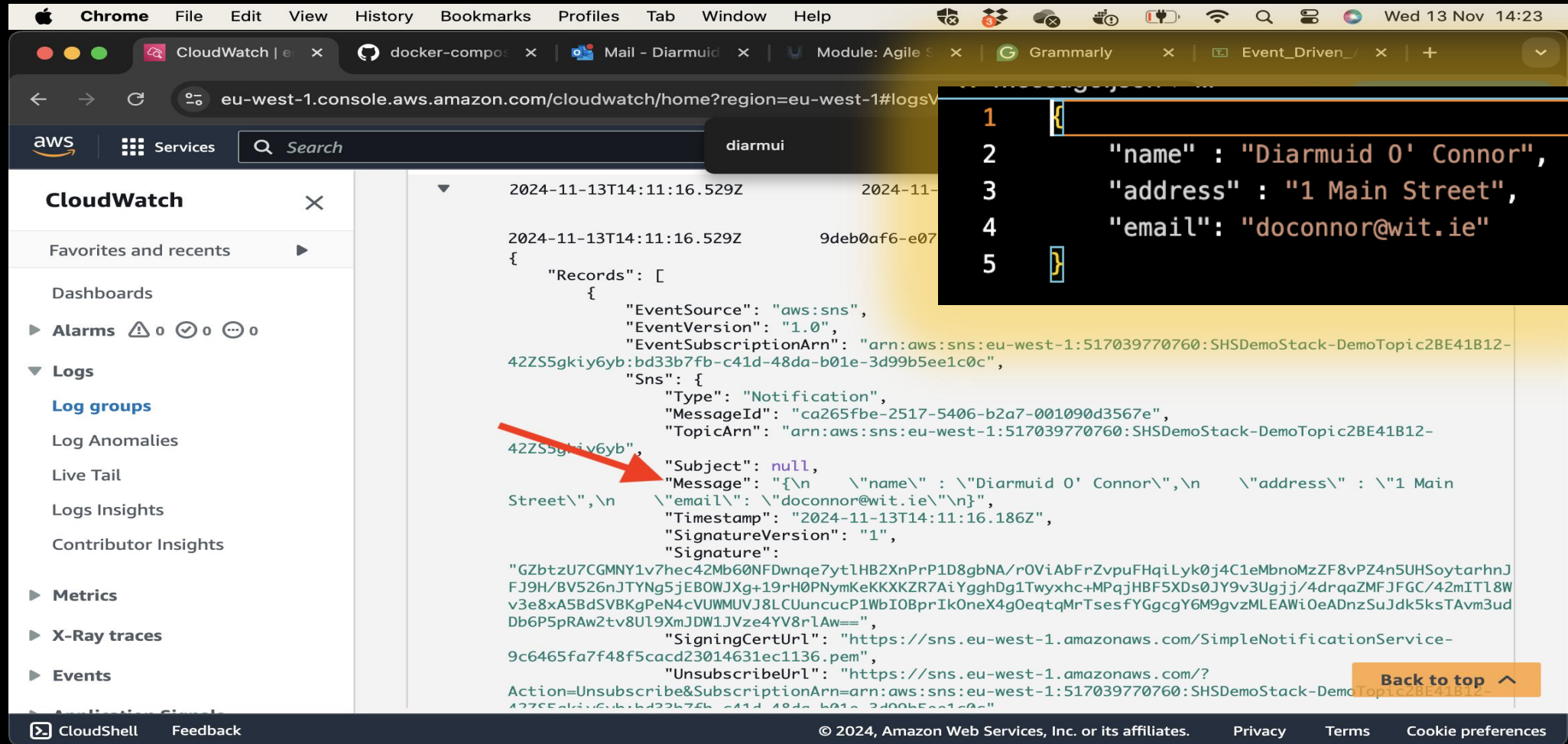
- Architecture:

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

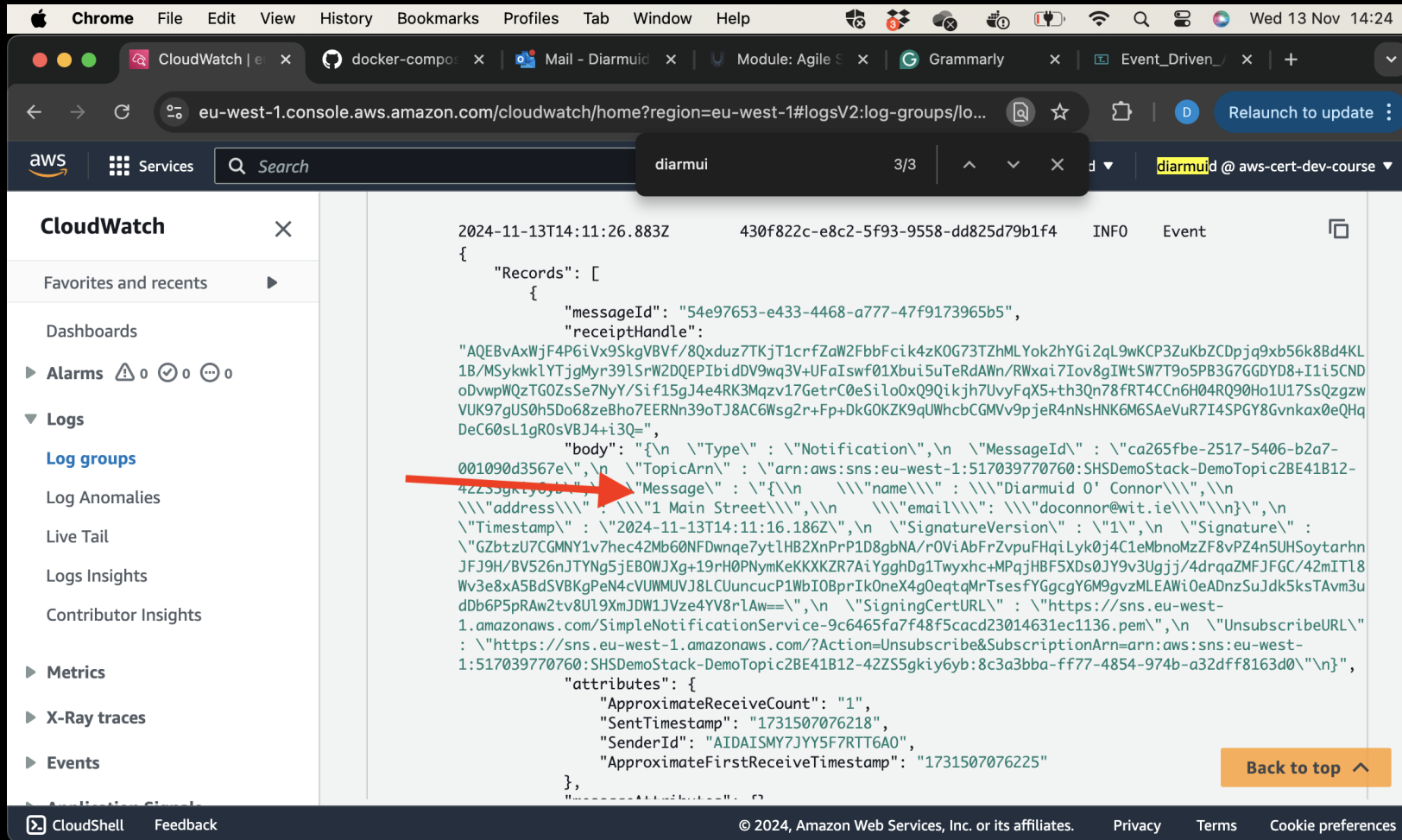


The screenshot shows the AWS CloudWatch console interface. The left sidebar contains navigation options: CloudWatch, Favorites and recents, Dashboards, Alarms, Logs, Log groups, Log Anomalies, Live Tail, Logs Insights, Contributor Insights, Metrics, X-Ray traces, and Events. The main content area displays a log group named 'diarmui' with a log entry from 2024-11-13T14:11:16.529Z. The log entry is a JSON object containing an 'EventSource' of 'aws:sns', an 'EventVersion' of '1.0', an 'EventSubscriptionArn' of 'arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-42Z55gkiy6yb:bd33b7fb-c41d-48da-b01e-3d99b5ee1c0c', and an 'Sns' object. The 'Sns' object contains a 'Type' of 'Notification', a 'MessageId' of 'ca265fbe-2517-5406-b2a7-001090d3567e', a 'TopicArn' of 'arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-42Z55gkiy6yb', a 'Subject' of null, and a 'Message' of '{\n \\'name\\\' : \\'Diarmuid O\' Connor\\\',\n \\'address\\\' : \\'1 Main Street\\\',\n \\'email\\\' : \\'doconnor@wit.ie\\\' }'. The 'Message' field is highlighted with a red arrow. A callout box on the right shows the JSON structure of the message, with the 'name', 'address', and 'email' fields highlighted.

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

Back to top

Demo – The Lambda Q consumer event parameter



The screenshot shows the AWS CloudWatch console interface. The left sidebar contains navigation options: CloudWatch, Favorites and recents, Dashboards, Alarms, Logs, Log groups, Log Anomalies, Live Tail, Logs Insights, Contributor Insights, Metrics, X-Ray traces, and Events. The main content area displays a log event from the 'diarmui' group. The event is dated 2024-11-13T14:11:26.883Z and has a message ID of 54e97653-e433-4468-a777-47f9173965b5. The message body is a JSON object containing a 'Type' of 'Notification', a 'MessageId' of 'ca265fbe-2517-5406-b2a7-001090d3567e', a 'TopicArn' of 'arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-42Z55gkty6yb', and a 'Message' field. The 'Message' field contains a JSON object with 'name' as 'Diarmuid O' Connor', 'address' as '1 Main Street', and 'email' as 'doconnor@wit.ie'. A red arrow points to the name 'Diarmuid O' Connor' in the message body. The event also includes a 'Timestamp' of '2024-11-13T14:11:16.186Z', a 'SignatureVersion' of '1', and a 'SigningCertURL' of 'https://sns.eu-west-1.amazonaws.com/SimpleNotificationService-9c6465fa7f48f5cadd23014631ec1136.pem'. The event attributes show an 'ApproximateReceiveCount' of '1', a 'SentTimestamp' of '1731507076218', a 'SenderId' of 'AIDAISMY7JYY5F7RTT6A0', and an 'ApproximateFirstReceiveTimestamp' of '1731507076225'. The bottom of the console shows the 'CloudShell' button and the footer text '© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences'.

```
2024-11-13T14:11:26.883Z      430f822c-e8c2-5f93-9558-dd825d79b1f4      INFO      Event      {
  "Records": [
    {
      "messageId": "54e97653-e433-4468-a777-47f9173965b5",
      "receiptHandle": "AQEBvAxWjF4P6iVx9SkgVBVf/8Qxdz7TKjT1crfZaW2FbbFcik4zKOG73TZhMLYok2hYGi2qL9wKCP3ZuKbZCDpj9xb56k8Bd4KL1B/MSykwkLYTjgMyr391SrW2DQEPiBdDV9wq3V+UFaIsfw01Xbui5uTeRdAWN/RWxai7Iov8gIwTSW7T9o5PB3G7GGDYD8+I1i5CNDodVwpWQzTG0ZsSe7NyY/Si f15gJ4e4RK3Mqzv17GetrC0eSiloXQ9Qikjh7UvyFqX5+th3Qn78fRT4CCn6H04RQ90Ho1U17SsQzgzWVUK97gUS0h5Do68zeBho7EERNn39oTJ8AC6Wsg2r+Fp+DkGOKZ9qUWhcbCGMVv9pjer4nNshNK6M6SAeVuR7I4SPGY8Gvnkax0eQHqDeC60sL1gR0sVBJ4+i3Q=",
      "body": "{\\n  \\\"Type\\\" : \\\"Notification\\\",\\n  \\\"MessageId\\\" : \\\"ca265fbe-2517-5406-b2a7-001090d3567e\\\",\\n  \\\"TopicArn\\\" : \\\"arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-42Z55gkty6yb\\\",\\n  \\\"Message\\\" : \\\"{\\n    \\\"name\\\" : \\\"Diarmuid O' Connor\\\",\\n    \\\"address\\\" : \\\"1 Main Street\\\",\\n    \\\"email\\\" : \\\"doconnor@wit.ie\\\"\\\"\\n}\\\",\\n  \\\"Timestamp\\\" : \\\"2024-11-13T14:11:16.186Z\\\",\\n  \\\"SignatureVersion\\\" : \\\"1\\\",\\n  \\\"Signature\\\" : \\\"GZbtzU7CGMNY1v7hec42Mb60NFDwnqe7ytLHB2XnPrP1D8gbNA/r0ViAbFrZvpvFHqLYk0j4C1eMbnMzZF8vPZ4n5UHSoytarhnJFJ9H/BV526nJTYNg5jEB0WJXg+19rH0PNymKeKKXZR7AiYgghDg1Twyxhc+MPqjHBF5XD0JY9v3Ugjj/4drqaZMFJFGC/42mITL8Wv3e8xA5BdSVBKgPeN4cVUWMUVJ8LCUuncucP1WbIOBprIkOneX4g0eqtqMrTsesfYGcgY6M9gvzMLEAWiOeADnzSuJdk5ksTAvm3udDb6P5pRAw2tv8U19XmJDW1JVze4YV8rLAW=\\\",\\n  \\\"SigningCertURL\\\" : \\\"https://sns.eu-west-1.amazonaws.com/SimpleNotificationService-9c6465fa7f48f5cadd23014631ec1136.pem\\\",\\n  \\\"UnsubscribeURL\\\" : \\\"https://sns.eu-west-1.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:eu-west-1:517039770760:SHSDemoStack-DemoTopic2BE41B12-42Z55gkiy6yb:8c3a3bba-f7f7-4854-974b-a32dff8163d0\\\"\\\"\\n}\\\",\\n  \\\"attributes\\\" : {\\n    \\\"ApproximateReceiveCount\\\" : \\\"1\\\",\\n    \\\"SentTimestamp\\\" : \\\"1731507076218\\\",\\n    \\\"SenderId\\\" : \\\"AIDAISMY7JYY5F7RTT6A0\\\",\\n    \\\"ApproximateFirstReceiveTimestamp\\\" : \\\"1731507076225\\\"\\n  }\\n}\\n"}",
      "attributes": {
        "ApproximateReceiveCount": "1",
        "SentTimestamp": "1731507076218",
        "SenderId": "AIDAISMY7JYY5F7RTT6A0",
        "ApproximateFirstReceiveTimestamp": "1731507076225"
      }
    }
  ]
}
```

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

Copy

```
{
  "Records": [
    {
      "messageId": "192a3ef6-176d-4ce9-a9fa-7f9994582a88",
      "receiptHandle": "AQEB8goKUGltRGemA3nrBYMBPQyFHTDH2JJA5u6hd+mwZ+RxsNu3IlszA9uKurF+uY3mHr8XnofiMJS2Zxlj
iy2nS6ohVV016mhlCwq64dla3JX1L+RIpcqXNh0F0qMzK56kF36BizSIxZS00XaviIiHx0xtrOFswp+u1n7hJ
+0TxBkW/V81c/b+jRdM6l1Hn7hqKb5V2xXkv/AJgfEo2sWdz5SVS8BLMDyIEGMMEngKq4Tnbpe0jiJpydUCX0
Z1zrEoaWoqrWxD0kX1P7fqQxZ73zGYkMHCvD/01bUIkKnywzgvTHDF2Fj2bCDNlsMRrq4qFDGoIAwx4V0Pl4
8ZP8DEpiYlnTFEEemA2AAK8oyDJ4AGNdp+lnidAWZiOGFau6Uj1uCW81bzjuFW0A1iOnRd1ndK0x6NosE/5xsY
leSyWZM="
```

SNS envelope

```
    "body": "{\n  \"Type\" : \"Notification\", \"MessageId\" : \"2625cdf8-
6f50-5eb3-aaaa-90195a3bce66\", \"TopicArn\" : \"arn:aws:sns:eu-west-
1:517039770760:SHSDemoStack-DemoTopic2BE41B12-paPr90UK7POD\", \"Message\" : \"
{\n  \"name\" : \"Diarmuid O' Connor\", \"address\" : \"1
Main Street\", \"email\" : \"doconnor@wit.ie\", \"
\"timestamp\" : \"2023-11-29T12:48:43.515Z\", \"SignatureVersion\" : \"1\", \"
\"Signature\" :
\"cW3s3KL5Jq1HBqhiabNrC3QEbXJZBR/g1b0C0QFk5eRkPKp2j8gGYkEGIsi0eerdgd+Pff9lo1M1NuGiYI7
0q3k9b0Fw9jkIh41+5tMnskDQk9mr/mdLHYFjIK2wmnMa7hggScsgWNfQtpLnt4Z8EGWrwa9lrNIRtLFHTkS
YuY/m9FdGwe3dDC8AYmsui8WzFP74vyPv46JkIgKDunqy4YsqUXdbCA2Hv7j/lV1WqXMKX21+6Hi8DF+u3q9l
lYzPWboTgbVlgWvzqbPFuY6tb6Z6yLEZi/ud00YitgiaoiWl8X9SEGqpnuo25+mIGgjM6AjVUGgbgWbiYU4wl
Mhvw==\", \"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-paPr90UK7POD:dd9b7420-ef0e-4e1b-88a2-e896b98d6612\"}\n\",
```

```
    "attributes": {
      "ApproximateReceiveCount": "1",
      "SentTimestamp": "1701262123540",
      "SenderId": "ATDATCMV71VVE7DTTGAQ"
```

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

Copy

{

"Records": [

No SNS envelope

{

"messageId": "973db4ca-53e2-4f29-ad3c-7bc252b02e25",

"receiptHandle":

"AQEBx/TenyP7WL3SsdKL7/QBif3japCb6NjIG0iLt+hDIXEvW0ps+2P05V7PFFx+Cgq/0lwUQJW6xWCJfgZ4

9feLyz5cxKBEwGL27kH1IY7roBcxgGDgbK/TbIcAbzVEcpoeCxmTdWCYZzvE66grzZ7jEVEDbnEkV6l0y+gXV

xeMtydar80E1989Hm5qBCrn7oG4T4FPamGZh907GOUJnVZPK8cSTtTNTATk8/HrcW

JYevV9Mpt3tvd00L0qk3rmOZZdzOPYRX0Ew7Uj9/D40jleOR+asKc0lsNetoJfqB1

FEH0Ik1yvp4WP0k9LOHGxkN0Timzg8yJzRrJL60ge3K3CRwopapt5w6giWYLRW+KTGm11s5+HgrDXrQFWL03

fDZoYPo=",

"body": "{\n \"name\": \"Diarmuid O' Connor\", \n \"address\":

\"1 Main Street\", \n \"email\": \"doconnor@wit.ie\" \n}",

"attributes": {

"ApproximateReceiveCount": "1",

"SentTimestamp": "1701262471745",

"SenderId": "AIDAISMY7JYY5F7RTT6A0",

"ApproximateFirstReceiveTimestamp": "1701262471751"

},

"messageAttributes": {},

"md5OfMessageAttributes": null,

"md5OfBody": "85f8fd703039e25159f4268695f0cd5f",

"..."

Back to top

Lambda Vs { SQS → Lambda } subscribers

- Disadvantages (Lambda subscriber)
 - No Batching is available when processing messages from SNS.
 - No control on Lambda Concurrency, messages are processed one by one as soon as they arrive.
 - Lambda function is responsible for handling errors/retries
 - Lambda DLQ needs to be handled separately.
- Advantages (Lambda subscriber)
 - Good for time-critical processing.

SNS - Delivery protocols and policies.

- SNS defines a delivery policy for each delivery protocol.
- The policy defines how SNS retries the delivery of messages when server-side errors occur (when the system that hosts the subscribed endpoint becomes unavailable). When the delivery policy is exhausted, SNS stops retrying the delivery 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 ¹	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					

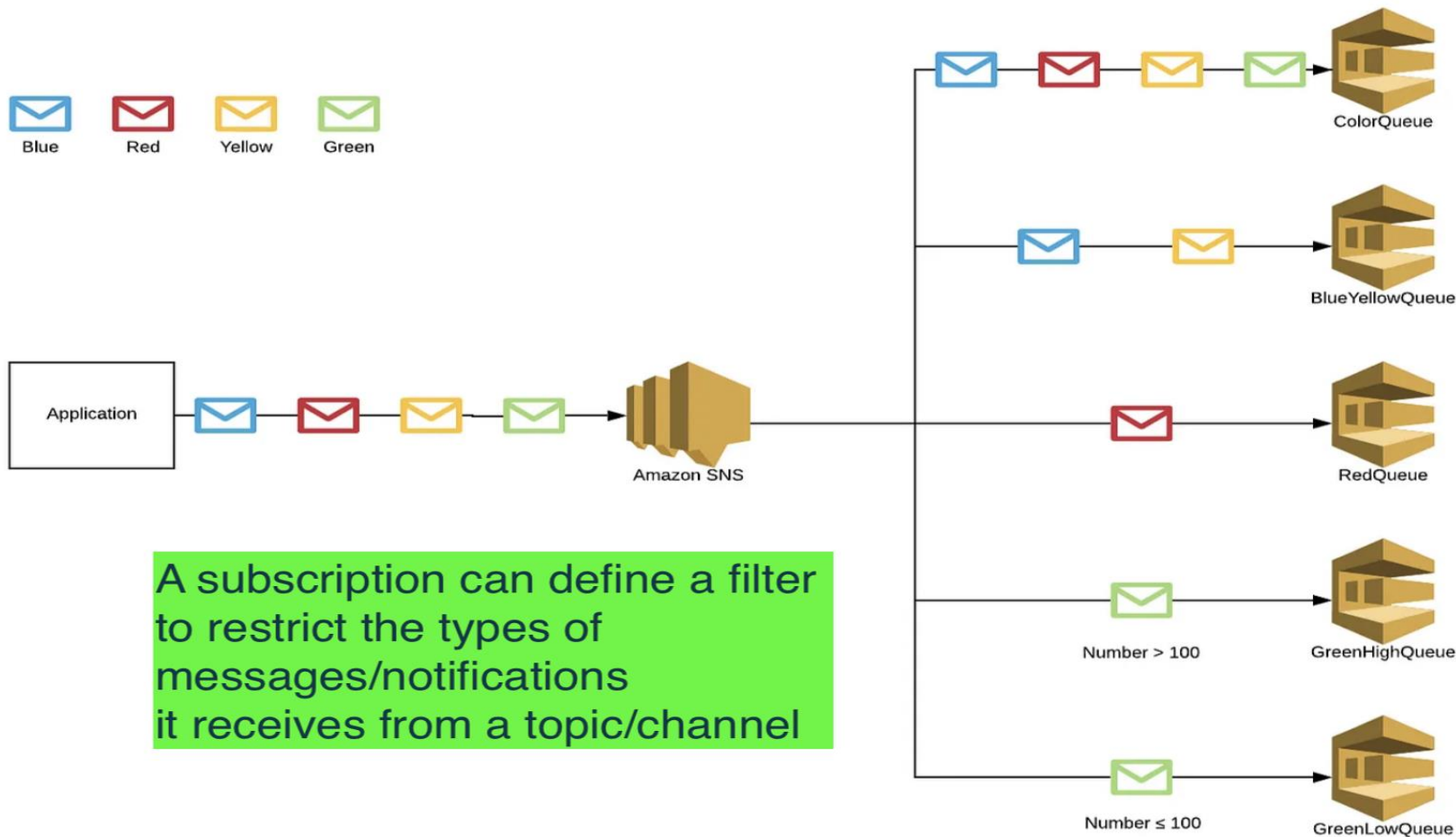
Lambda subscriber DLQ

- 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 (Sub\
|
| → DLQ → Lambda (Consumer)

Fan Out pattern - Filtering.



Filtering

- Filtering policy can be based on message attributes or the message body.
 - Filtering criteria:
 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)

- The Lambda subscription only concerns messages with the attribute `user_type` set to Student or Lecturer.

Demo – Message Attribute Filtering


```
demoTopic.addSubscription(  
    new subs.LambdaSubscription(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 Street\",\n      \"email\" : \"doconnor@wit.ie\"\n    }",  
        "Timestamp": "2024-11-19T13:44:32.548Z",  
        "SignatureVersion": "1",  
        "Signature":  
        "c0b04CTqDRbpBBnIi6+SMv1CAD6pLXuR/oWrLOEUjR5sUg80LIk66M8G+o2Qr5N32Dv2o6jdXLhiq27r5KbQCD6YknrDjXCm2CHUZ5FI1E4dYPqWgULehNAIRThOgpOAzilKLbAJZhpKmfpartMaAZn0iW3CqolPXTJjiEp1yxEsAuo4yldsmEDvrkzHE+A9r/ZpLW23W5kLWPhLB8+0uoI2bYC3prUiM6UXHxi9/hF4EI7HwLoL8IPTu7//6KtAkrIWUe41P1nDvtxDfUNKxVRn8HPIIz/woUj09rJeXdnRrhVgYdKRoji53UqIBcm4drhI0FF968spmpa25LFwQ==",  
        "SigningCertUrl": "https://sns.eu-west-1.amazonaws.com/SimpleNotificationService-9c6465fa7f18f5cacd23014631ec1136.pem",  
        "UnsubscribeUrl": "https://sns.eu-west-1.amazonaws.com/?Action=Unsubscribe&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"  
          }  
        }  
      }  
    ]  
  }  
}
```



Message Body Filtering

- Filtering based on properties of the message body.
 - Body is a JSON structure.
- Same filtering criteria as attribute filtering
- Demo – Architecture:
S3 (Pub) → SNS --<Filter>→ SQS (Sub) → Lambda fn (Con)

The filter only allows messages for objects named 'image*' uploaded to an S3 bucket.

Demo - Understand the message structure

```
▼ 2023-12-06T09:57:47.957+00:00 2023-12-06T09:57:47.957Z ee6e2815-00d4-5250-814d-75ff4711ad INFO Event {
  Records: [
    {
      messageId: '5373d063-4eef-4121-bdcB-faa7746366b7',
      receiptHandle:
        'AQEB+IHF20Vjb7XyGgCjqFX0xd0rgGJZCPgjt1bNjqLYVSnE9HmroL0KIBe12K7HSmAuleV9aHWG+LmH57uhHpnGr
        A6ZZNMUx3nKU6JUj0DveMQOLVvaFdF27HftD/I78yGdjTL29Z+CskZ2ZHD0EwaF4DjQSzhmRz/MeNYgXGoVwgHgGSwaO
        cR6BCjkAen4fw25Dd9BdkShnB+MbRVK+g1Kc9DKQcisCJSRxtc0Z22/uEEErLa0F10SY0ouyZbUFEVjxotaQIhfRA0
        RjZF/0',
      body: '{"Records":[{"eventVersion":"2.1","eventSource":"aws:s3","awsRegion":"eu-west-1","eventTime":"2023-12-
        06T09:57:26.212Z","eventName":"ObjectCreated:Put","userIdentity":{"principalId":"AWS:AIDAXQYPYZSEFH75QIS7P"},"requestParameters":
        {"sourceIPAddress":"193.1.184.238"},"responseElements":{"x-amz-request-id":"JPZP22KC2MG3FN13","x-amz-id-
        2":"NU608qNARI7gUJLaUXJhnmoRErDtBHsPljXJj0jrjMtds0LsQwnbqg8sEKrTAe6itXbNdWtXp7Um9Ko9gF0DRNwH2P9ZUzC2"},"s3":
        {"s3SchemaVersion":"1.0","configurationId":"NmY5NjRmYmMtYjAyOC00YWZjLWlYMGItYTRmZDNLNGRmN2My","bucket":{"name":"edastack-images9bf4dcd5-
        nfqrecrehdv5"},"ownerIdentity":{"principalId":"A1K7SN8AC8I6PY"},"arn":"arn:aws:s3:::edastack-images9bf4dcd5-nfqrecrehdv5"},"object":
        {"key":"image2.jpeg","size":237793,"eTag":"75609d041989d92cfc585fa330ddcb6d","sequencer":"006570458615B86BFC"}}}]}' ,
      attributes: [Object],
      messageAttributes: {},
      md5OfBody: 'a836412dcbebede8748b5e7c144ad1c5',
      md5OfMessageAttributes: null,
      eventSource: 'aws:sqs',
      eventSourceARN: 'arn:aws:sqs:eu-west-1:517039770760:EDASack-imgcreatedqueueB98FF37D-HCKw0QKF7JN5',
      awsRegion: 'eu-west-1'
    }
  ]
}
```

- Structure of event passed to lambda function for an SQS Q poll operation.
- The message body is the S3 notification sent to SNS - in stringified form.

Demo - Understand the message structure

```
2023-12-06T09:57:47.965Z    ee6e2815-00d4-5250-814d-75ff471711ad    INFO    bodys
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "eu-west-1",
      "eventTime": "2023-12-06T09:57:26.212Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AIDAXQYPYZSEFH75QIS7P"
      },
      "requestParameters": {
        "sourceIPAddress": "193.1.184.238"
      },
      "responseElements": {
        "x-amz-request-id": "JPZP22KC2MG3FN13",
        "x-amz-id-2": "NU608qNARI7gUJLaUXJhnmoRErDtBHsPljXJj0jrjMtdsOLsQwnbqg8sEKrTAe6itXbNdWtXp7Um9Ko9gF0DRNwH2P9ZUzC2"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "NmY5NjRmYmMtYjAyOC00YWZjLWlYMGItYTRmZDNlNGRmN2My",
        "bucket": {
          "name": "edastack-images9bf4dcd5-nfqrecrehdv5",
          "ownerIdentity": {
            "principalId": "A1K7SN8AC8I6PY"
          },
          "arn": "arn:aws:s3:::edastack-images9bf4dcd5-nfqrecrehdv5"
        },
        "object": {
          "key": "image2.jpeg",
          "size": 237793,
          "eTag": "75609d041989d92cfc585fa330ddcb6d",
          "sequencer": "006570458615B86BFC"
        }
      }
    }
  ]
}
```

Copy

- Parsed form of S3 notification sent to SNS.
- This structure is used to define the subscription body filter

Demo – Filter Policy

```
newImageTopic.addSubscription(  
    new subs.SqsSubscription(imageProcessQueue, {  
        filterPolicyWithMessageBody: {  
            Records: sns.FilterOrPolicy.policy({  
                s3: sns.FilterOrPolicy.policy({  
                    object: sns.FilterOrPolicy.policy({  
                        key: sns.FilterOrPolicy.filter(  
                            sns.SubscriptionFilter.stringFilter({  
                                matchPrefixes: ["image"],  
                            })  
                        })  
                    },  
                },  
            },  
        },  
        rawMessageDelivery: true,  
    })  
);
```

Demo – Filter Policy

- Ex. 2 Only allow messages with the S3 schema version of 2.0.

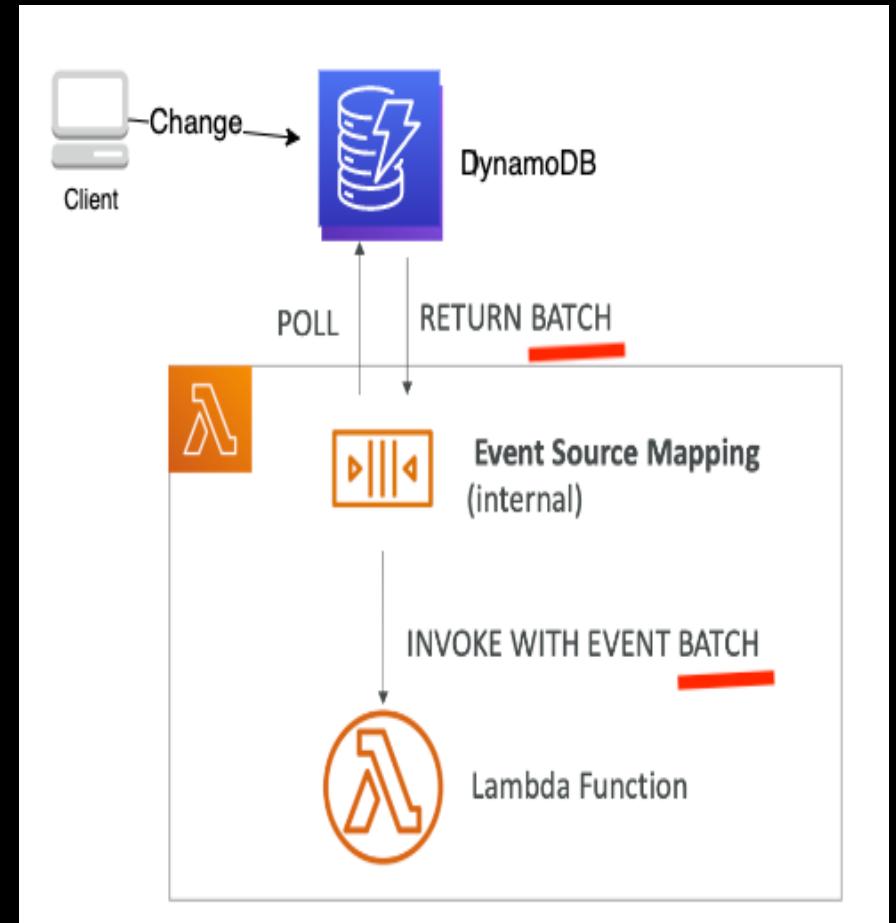
```
92     newImageTopic.addSubscription(  
93         new subs.SqsSubscription(imageProcessQueue, {  
94             filterPolicyWithMessageBody: {  
95                 Records: sns.FilterOrPolicy.policy({  
96                     s3: sns.FilterOrPolicy.policy({  
97                         s3SchemaVersion: sns.FilterOrPolicy.filter(  
98                             sns.SubscriptionFilter.stringFilter({  
99                                 allowlist: ["2.0"],  
100                             })  
101                     }),  
102                 }),  
103             }),  
104         },  
105         rawMessageDelivery: true,  
106     })  
107 );
```

Reminder (Why)

- It's better to decouple an app's compute components by using a range of AWS services/techniques:
 - SQS: queue model.
 - SNS: pub/sub/subscribe model.
 - Data streams.
 - Triggers.
- These techniques result in:
 - Reduces latency; Increase availability; Reduces complexity (by decreasing dependency).

Lambda – Event source mappings

- Sources: DynamoDB streams, SQS, Kinesis streams.
- Common denominator: events are polled from the source.
- Your Lambda function is invoked synchronously by the Lambda service. The events are placed in an Event Queue.
- Lambda service retries on errors. 3 tries with exponential backoff



DynamoDB streams

- Declare the stream in the Table.
- Declare the stream as an event source (trigger) for the lambda function.
- Tip: console.log the event in the function to see its structure

```
const myTable = new Table(this, "MyTable", {
  billingMode: BillingMode.PAY_PER_REQUEST,
  partitionKey: { name: "ID", type: AttributeType.STRING },
  removalPolicy: RemovalPolicy.DESTROY,
  tableName: "MyTable",
  stream: StreamViewType.NEW_IMAGE,
});

// Create source mapping between stream and lambda function
processDataFn.addEventSource(
  new DynamoEventSource(reviewsTable, {
    startingPosition: StartingPosition.LATEST,
  })
);
```



The diagram consists of two red arrows. The first arrow originates from the 'stream: StreamViewType.NEW_IMAGE' property in the Table constructor and points to the 'processDataFn.addEventSource' method call. The second arrow originates from the 'processDataFn.addEventSource' method call and points to the 'DynamoEventSource' constructor, illustrating the flow of data from the table's stream to the lambda function's event source.

DynamoDB stream types

- DynamoDB stream type controls what is written to the stream.
 - NEW_IMAGE – The entire table item, after it was modified.
 - NEW_AND_OLD_IMAGES - Both the pre- and post-modification states of the item.
 - OLD_IMAGE - The tableitem, before it was modified
 - KEYS_ONLY - Only the key attributes of the modified item

```
const myTable = new Table(this, "MyTable", {
  billingMode: BillingMode.PAY_PER_REQUEST,
  partitionKey: { name: "ID", type: AttributeType.STRING },
  removalPolicy: RemovalPolicy.DESTROY,
  tableName: "MyTable",
  stream: StreamViewType.  
});  
  
// Create source mapping  
processDataFn.addEventSource(  
  new DynamoEventSource(reviewtable, {
```

You, 1 second ago • Uncommitted

- KEYS_ONLY (enum member) StreamViewType
- NEW_AND_OLD_IMAGES
- NEW_IMAGE
- OLD_IMAGE

DynamoDB stream types

- The event object passed to the lambda function includes information on the type of table operation associated with the stream entry, i.e. insert, update, delete.
 - Use `console.log` (and JSON `parse/stringify`) to examine the structure of a stream event.