Developing Solutions with Amazon SQS and Amazon SNS

Module objectives

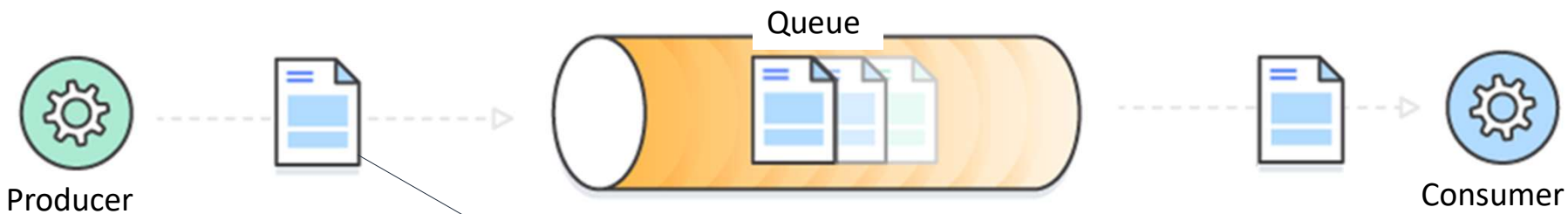
- Recall how message queues work
- Describe Amazon SQS
- Send messages to an SQS queue
- Describe Amazon SNS
- Explain Amazon SNS concepts

Part 1: Introduction to message queues

Message queues

Producer = Application component that produces messages and adds them to queue

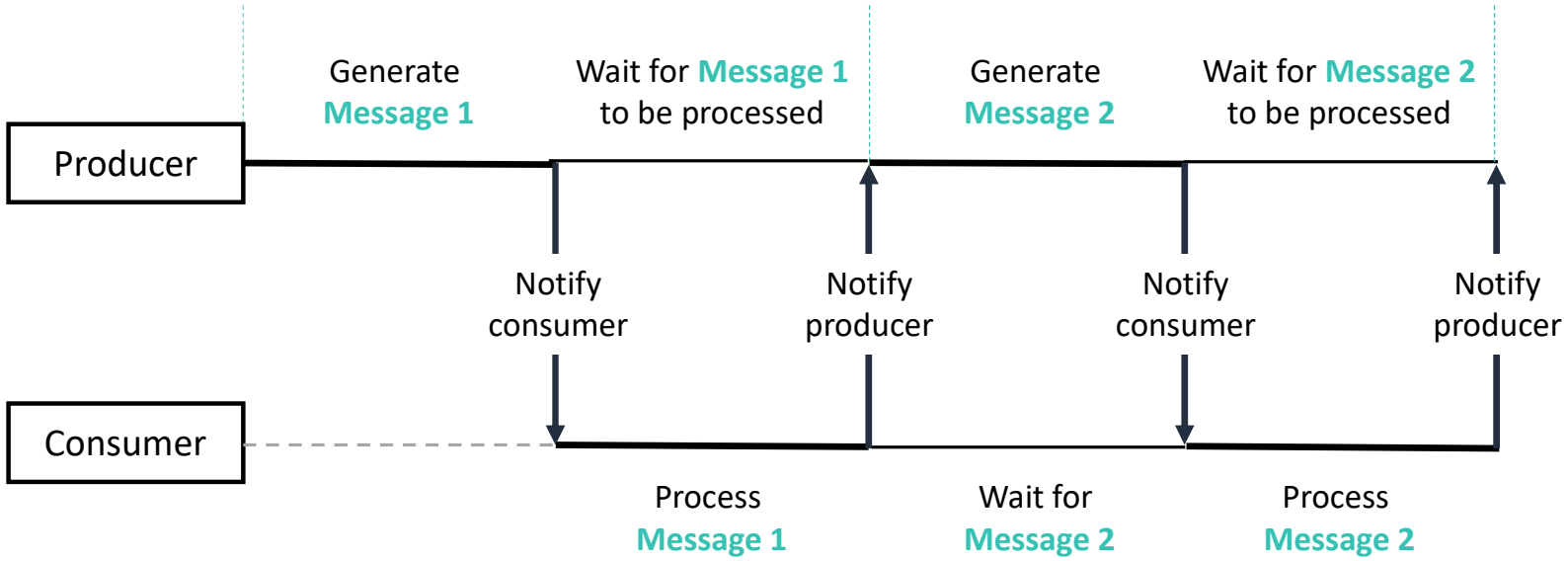
Consumer = Application component polls queue for messages and processes them



A **message** is for communication between software components—not between people (that is, it's not an email or text message).

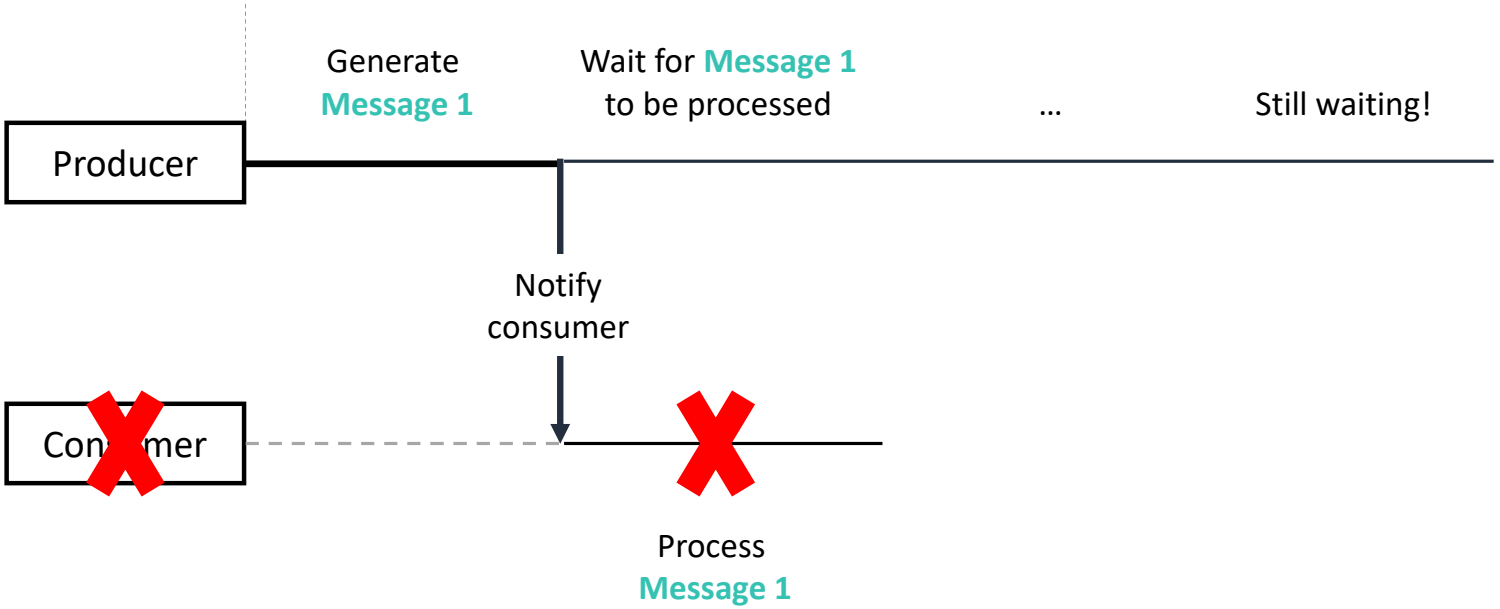
Pull mechanism

Synchronous process



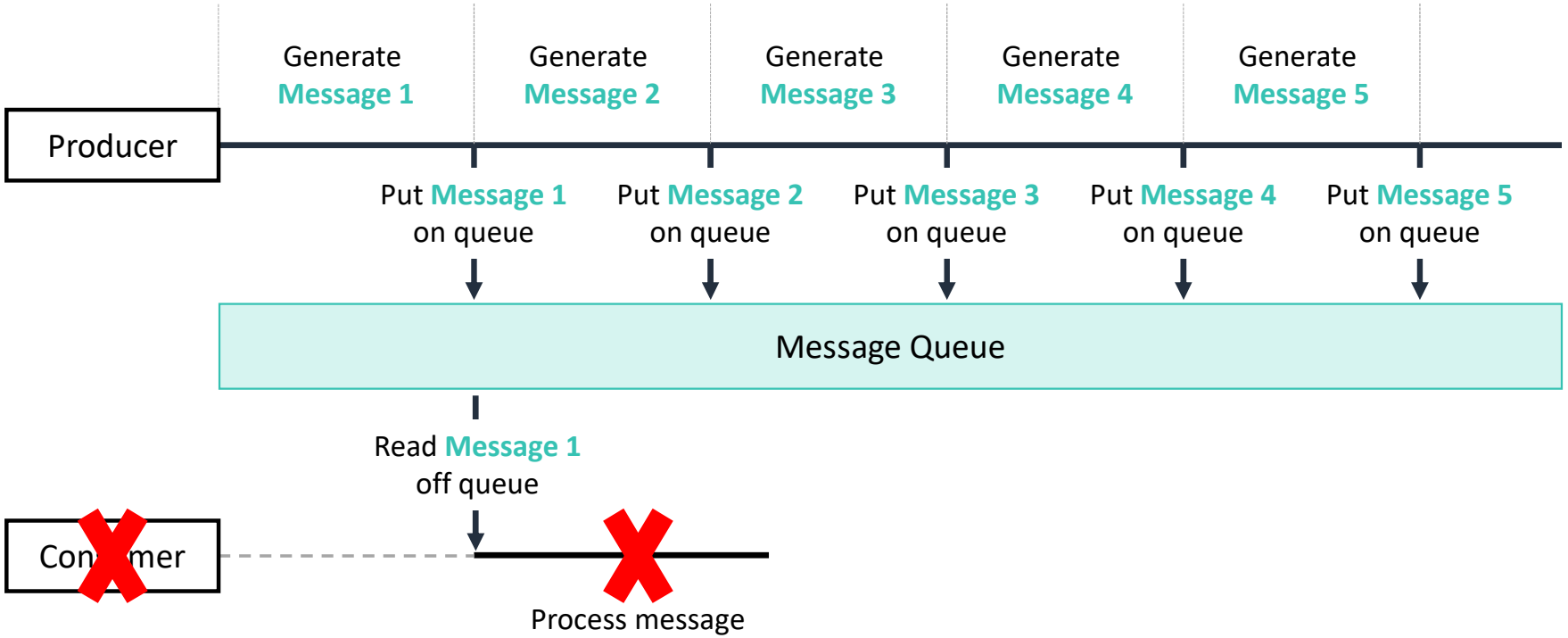
Disadvantage of a synchronous process

Synchronous = Tightly coupled system



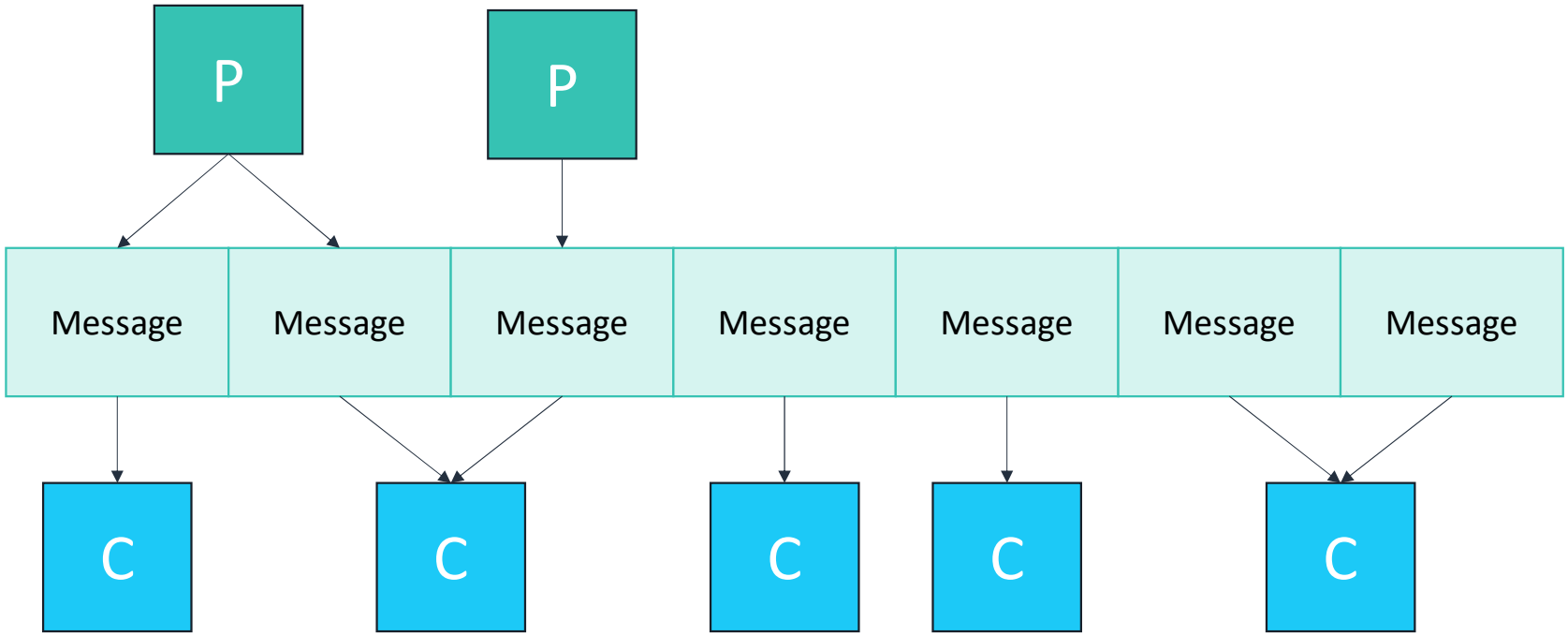
Asynchronous process

Asynchronous = Loosely coupled system



An asynchronous process is scalable

Producers



Consumers

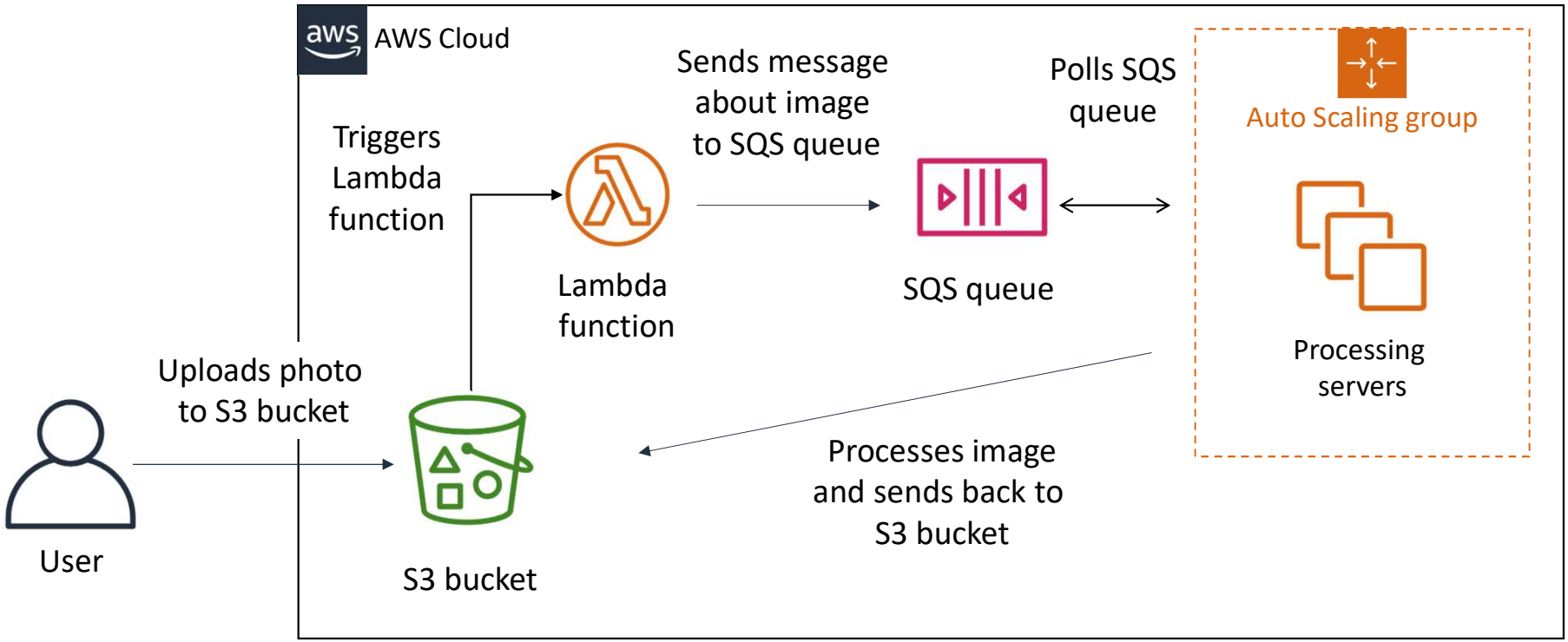
Part 2: Introduction to Amazon SQS

Amazon Simple Queue Service

Amazon SQS

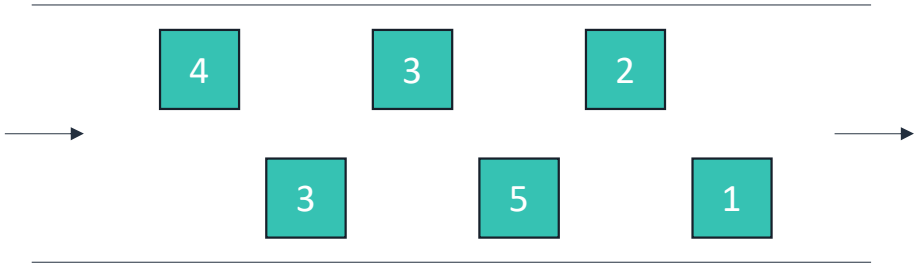
- Fully managed message queuing service
- Delivers messages reliably
- Scales elastically
- Cost-effective
- Keeps sensitive data secure

Amazon SQS architecture example



Amazon SQS queue types

Standard queue



- Message ordering is not guaranteed
- Messages might be duplicated
- High throughput

FIFO queue

- Message ordering is preserved
- Messages are only received once
- Limited throughput (300 transactions per second, per action)

Amazon SQS queue type use cases

Standard queue

- Upload media while resizing or encoding it
- Process a high number of credit card validation requests
- Schedule multiple entries to be added to a database

FIFO queue

- Bank transactions
- Credit card transactions
- Course enrollment

Part 3: Amazon SQS developer concepts

Amazon SQS message lifecycle

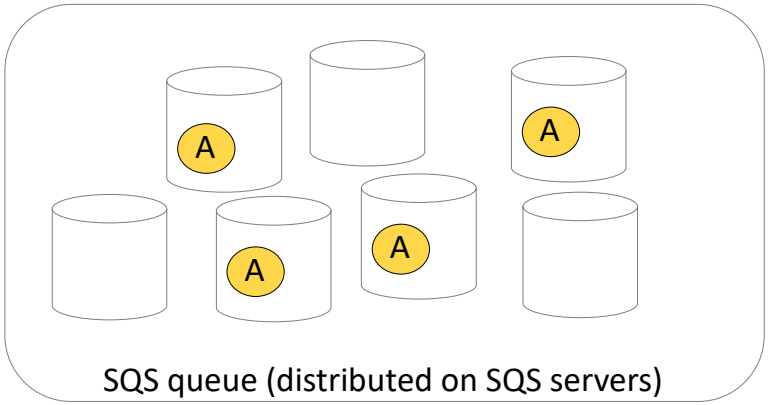
Send message

- Send message
- Receive message
- Delete message

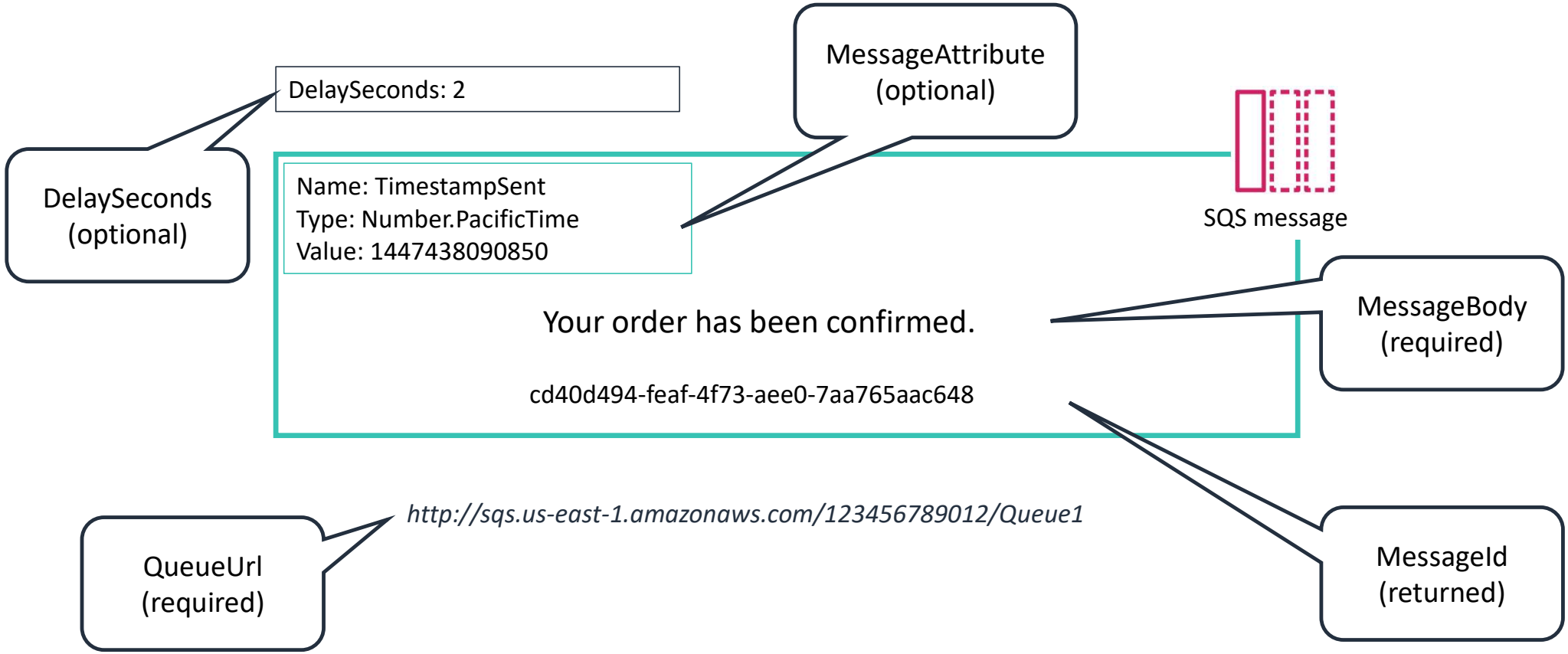
Producer

Producer sends Message A to the SQS queue (SendMessage operation)

A



SendMessage operation



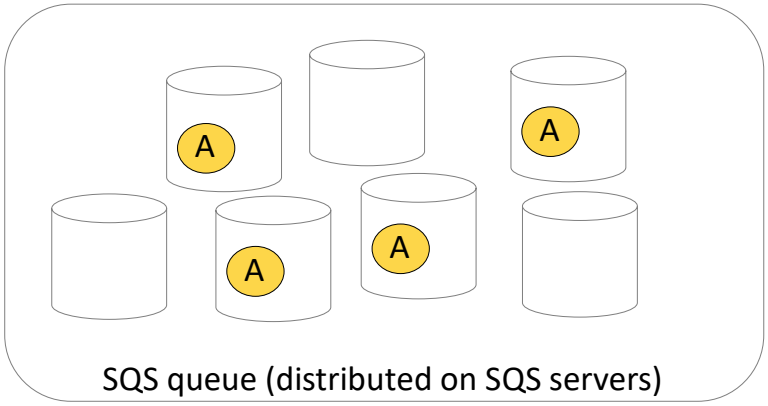
Receive message

- Send message
- Receive message
- Delete message

Consumer

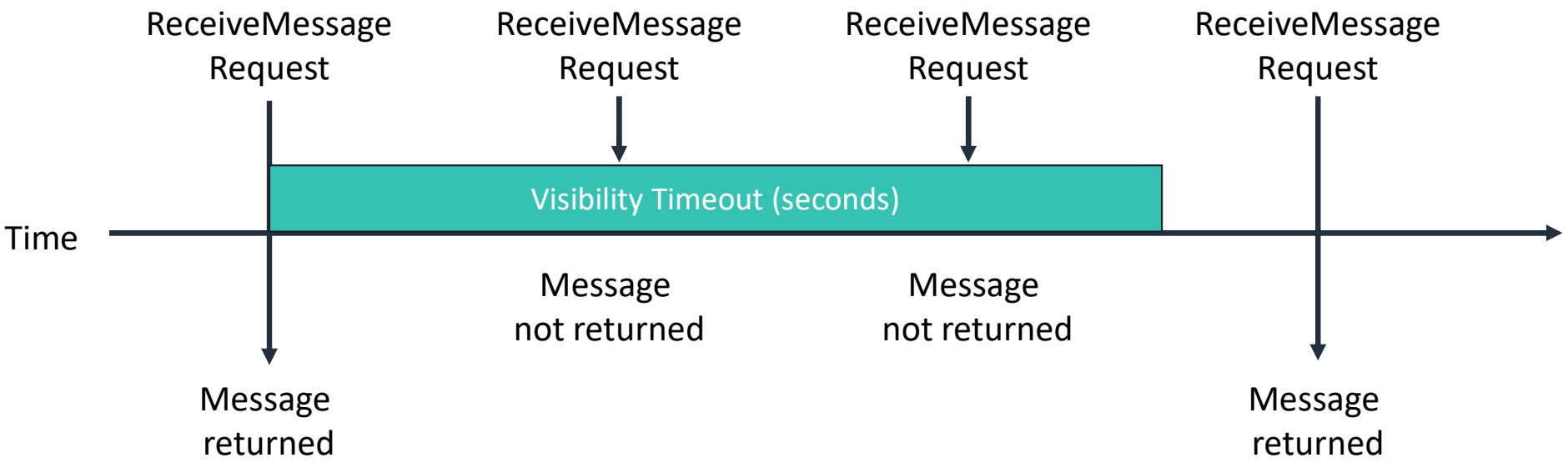
Consumer retrieves Message A and receives receipt handle. Processes message. (ReceiveMessage operation)

A **+ receipt handle**



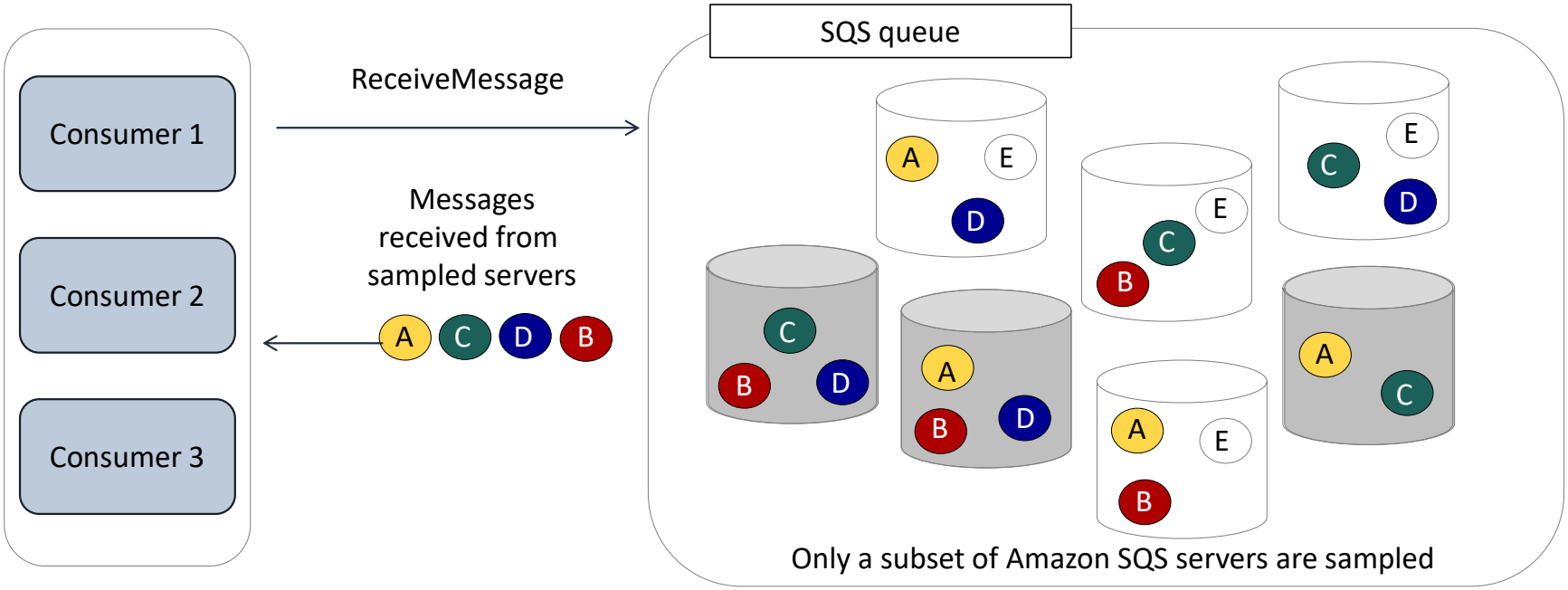
Visibility timeout

VisibilityTimeout = value between 0 seconds and 12 hours



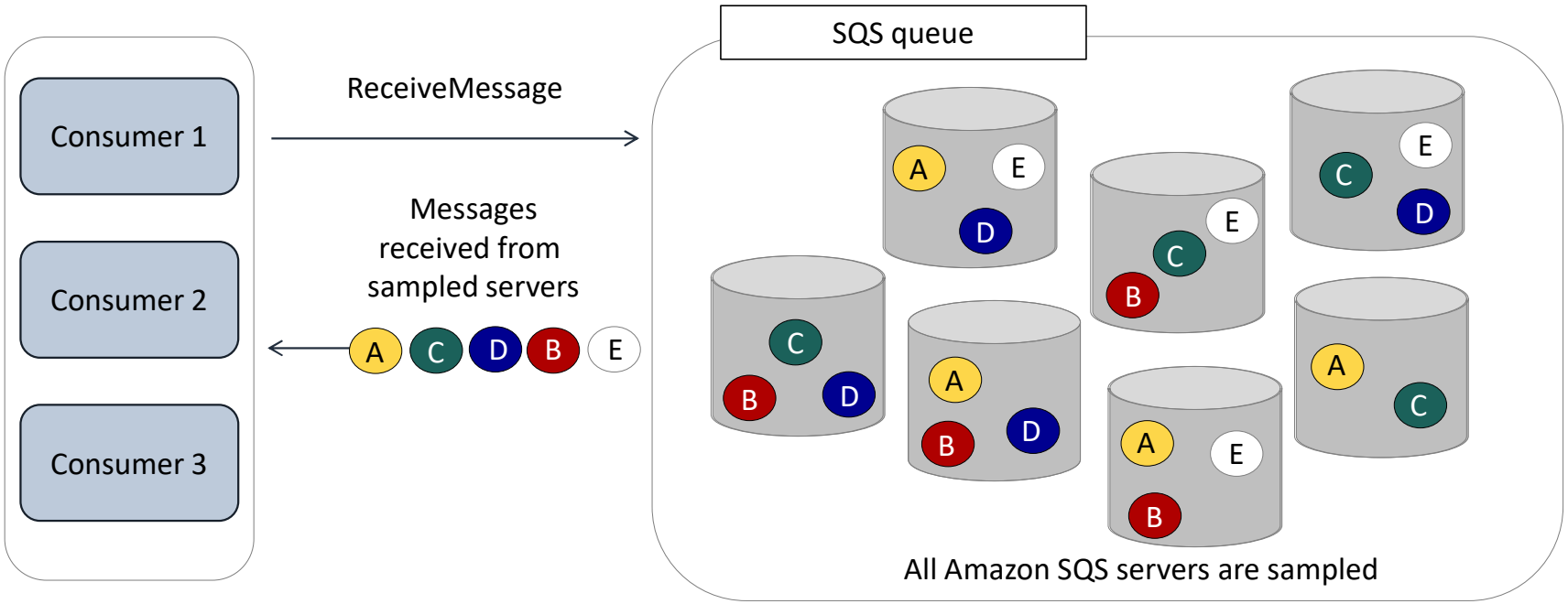
Short polling

WaitTimeSeconds = 0 seconds



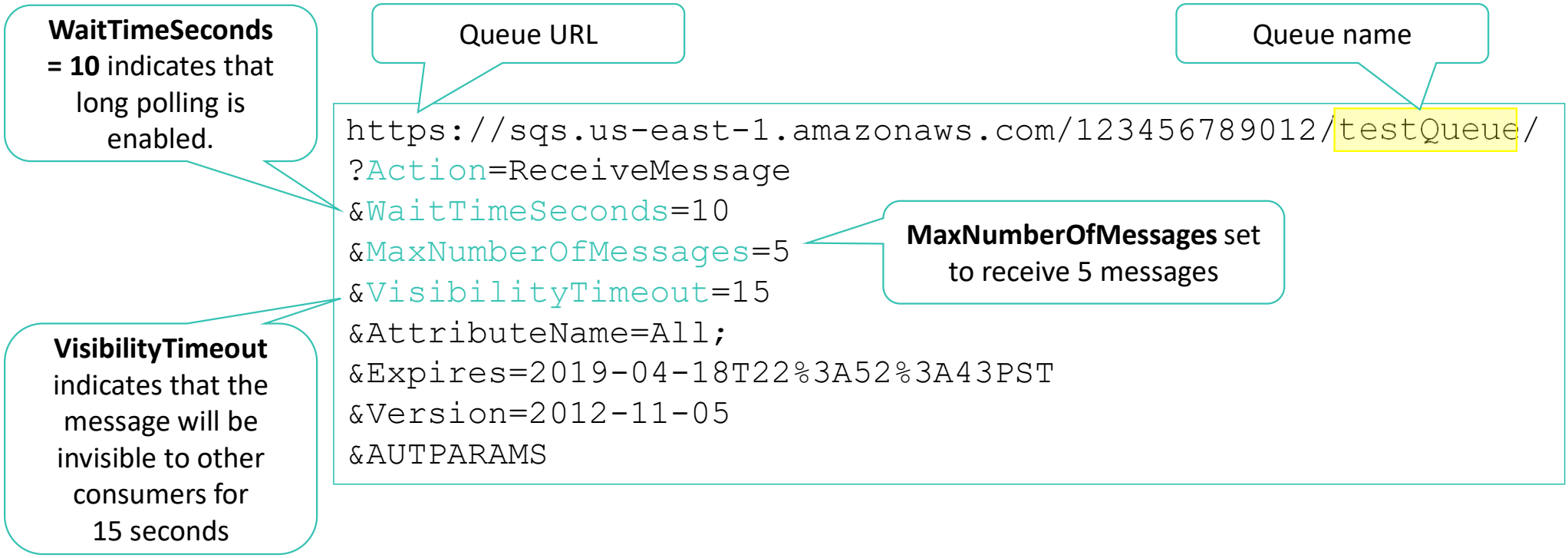
Long polling

WaitTimeSeconds = Value between 1 and 20 seconds



Long polling helps reduce cost and network traffic.

ReceiveMessage example



Delete message

- Send message
- Receive message
- Delete message

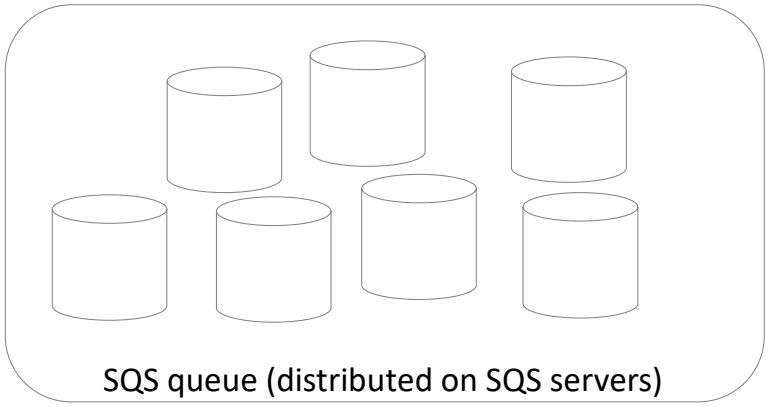
Amazon SQS automatically deletes messages that have been in the queue for longer than the **message retention period**.

Consumer processes Message A and deletes it from the queue during the visibility timeout period

Consumer

A

DeleteMessage operation



SQS queue (distributed on SQS servers)

DeleteMessage example

Queue URL of queue to delete message from

Queue name

```
https://sqs.us-east-1.amazonaws.com/123456789012/testQueue/
?Action=DeleteMessage
&ReceiptHandle=MbZj6wDWli%2BJvwwJaBV%2B3dcjk2YW2vA3%2BSTFFljT
M8tJJg6HRG6PYSasuWXPJB%2BCwLj1FjgXUv1uSj1gUPAWV66FU/WeR4mq2OKpEGY
WbnLmpRCJVAyeMjeU5ZBdtcQ%2BQEauMZc8ZRv37sIW2iJKq3M9MFx1YvV11A2x/K
SbkJ0=
&Version=2012-11-05
&Expires=2012-04-18T22%3A52%3A43PST
```

ReceiptHandle is the receipt handle associated with the message to delete.

Dead-letter queues

- Queue of messages that could not be processed
- Use with standard queues
- Use to help troubleshoot incorrect message transmission operations

Amazon SQS queue operations

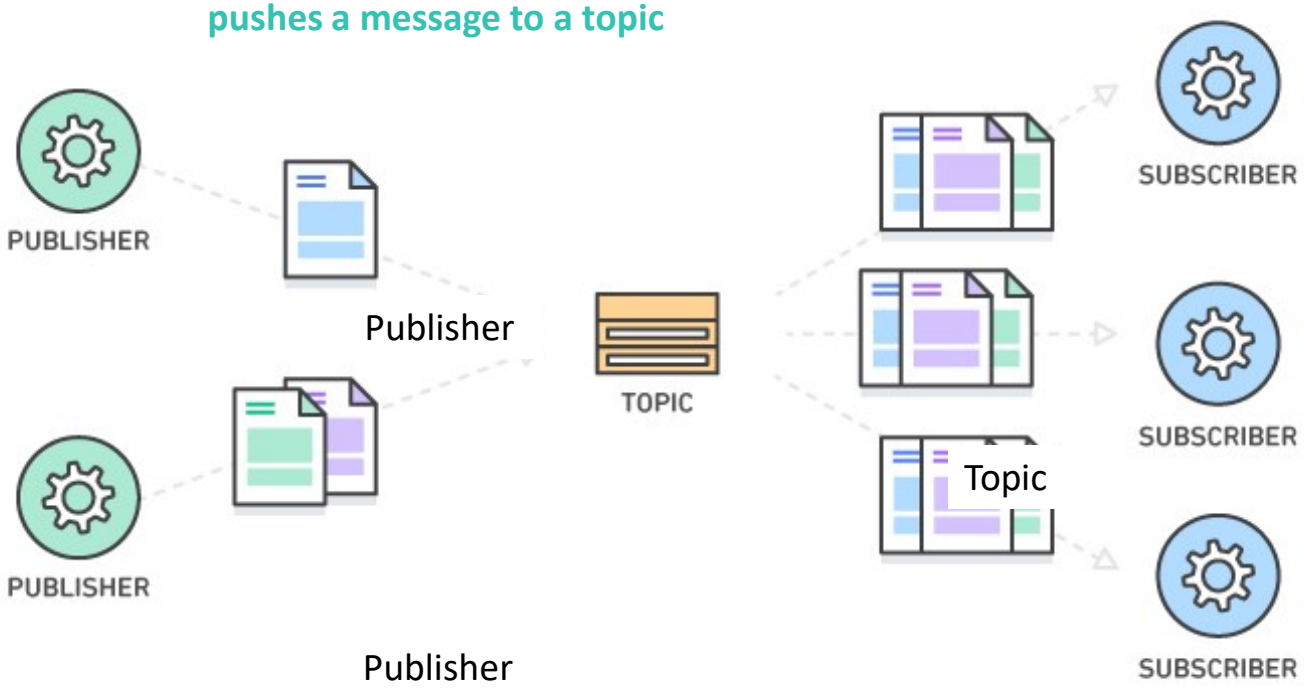
- `CreateQueue`
  - **Attributes:** `DelaySeconds, MaximumMessageSize, MessageRetentionPeriod, ReceiveMessageWaitTimeSeconds, VisibilityTimeout`
- `SetQueueAttributes`
- `GetQueueAttributes`
- `GetQueueUrl`
- `ListQueues`
- `DeleteQueue`

Part 4: Introduction to Amazon SNS

Pub/sub messaging

Publisher = Component that pushes a message to a topic

Subscriber = Component that subscribes to a topic



Publisher

Publisher

Subscriber

Subscriber

Subscriber

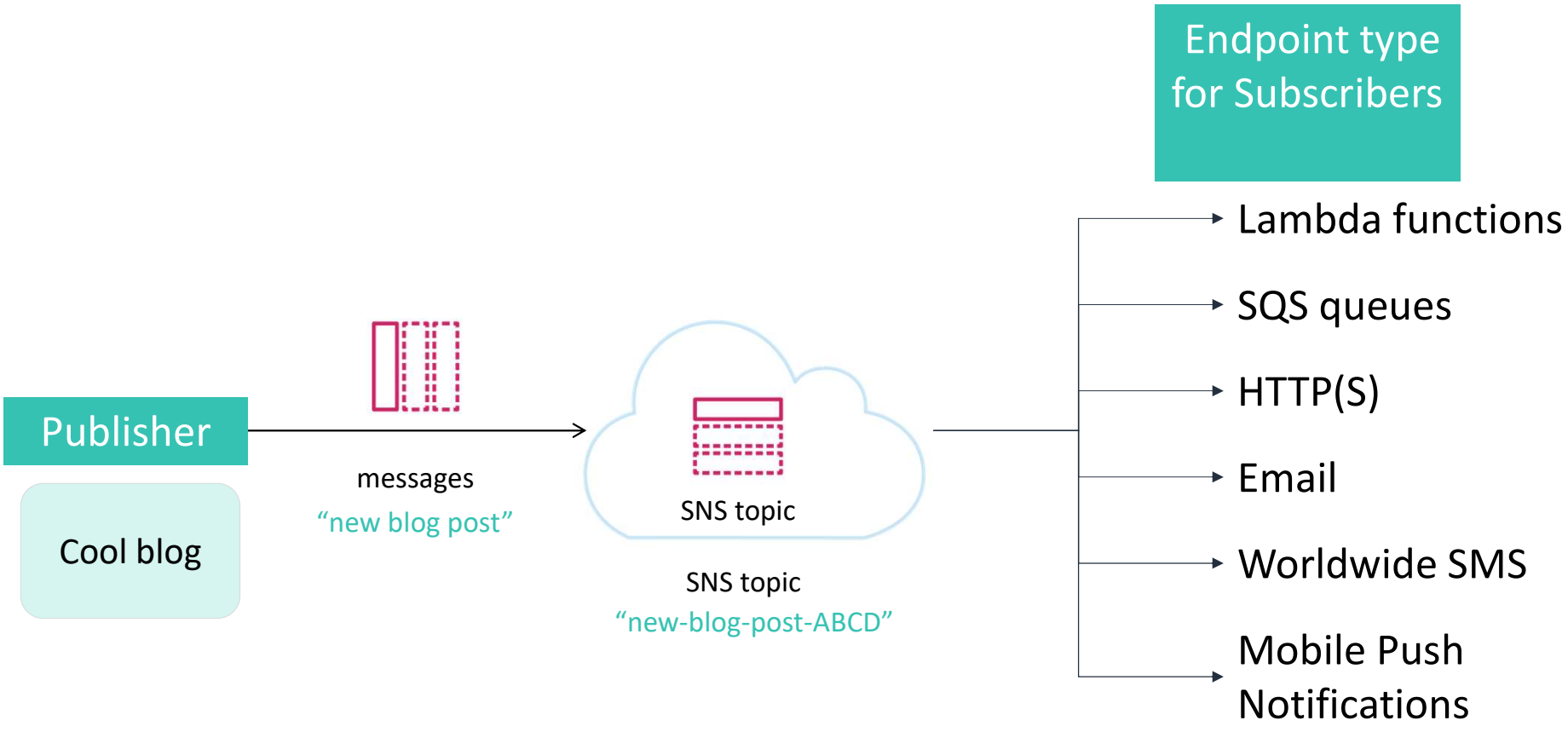
Push mechanism

Amazon Simple Notification Service

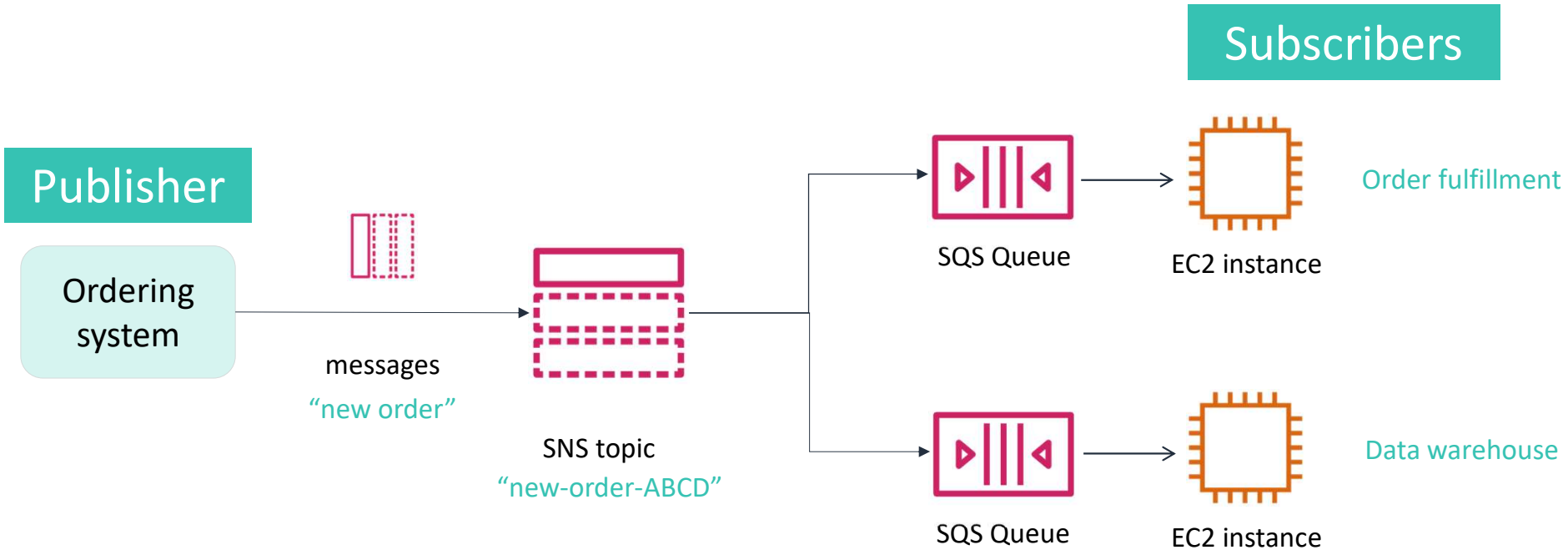
Amazon SNS

- Highly available, durable, secure, and fully managed pub/sub messaging and mobile communications service
- Decouples microservices, distributed systems, and serverless applications

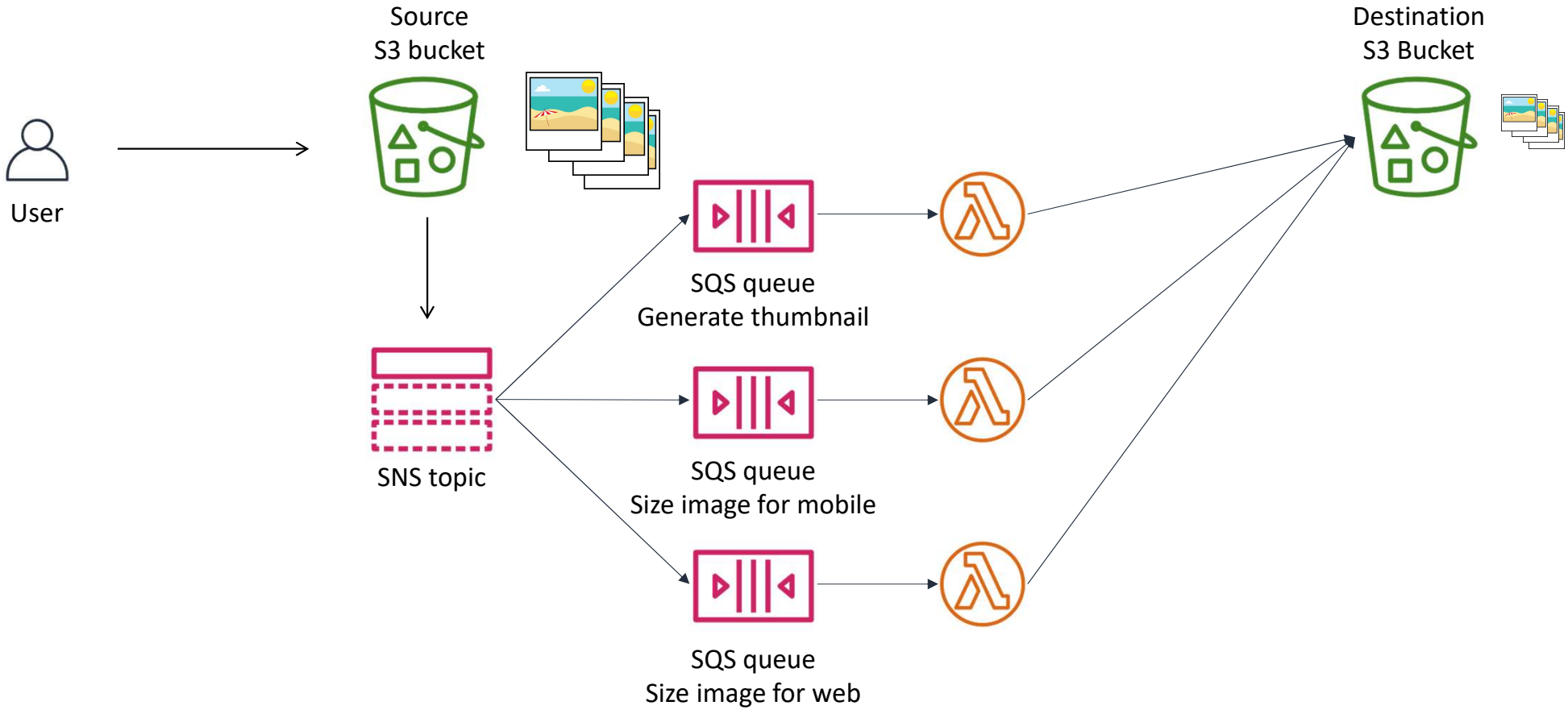
Amazon SNS for pub/sub messaging



Use case: Fanout design pattern



Fanout example: Image processing



Part 5: Amazon SNS developer concepts

Amazon SNS operations

CreateTopic

- Input: Topic name
- Output: ARN of topic

Subscribe

- Input:
  - Subscriber's endpoint
  - Protocol
  - ARN of topic

ConfirmSubscription

- Input: Token sent to endpoint
- Output: ARN of topic

DeleteTopic

- Input: ARN of topic

Publish

- Input:
  - Message
  - Message attributes (optional)
  - Message structure:json (optional)
  - Subject (optional)
  - ARN of topic
- Output:
  - Message ID

Amazon SNS raw message delivery

```
{
        "Type" : "Notification",
        "MessageId" : "63a3f6b6-d533-4a47-aef9-fcf5cf758c76",
        "TopicArn" : "arn:aws:sns:us-west-2:123456789012:MyTopic",
        "Subject" : "Testing publish to subscribed queues",
        "Message" : "New order!",
        "Timestamp" : "2012-03-29T05:12:16.901Z",
        "SignatureVersion" : "1",
        "Signature" : "EXAMPLEnTrFPa37tnVO0FF9Iau3MGzjlJLRfySEoWz4uZHSj6ycK4ph71Zm
dv0NtJ4dC/Vz20zxmF9b88R8GtqjfKB5woZZmz87HiM6CYDTo3l7LMwFT4VU7ELtyaBBafhPTg9O5CnKkg=",
        ...
}
```

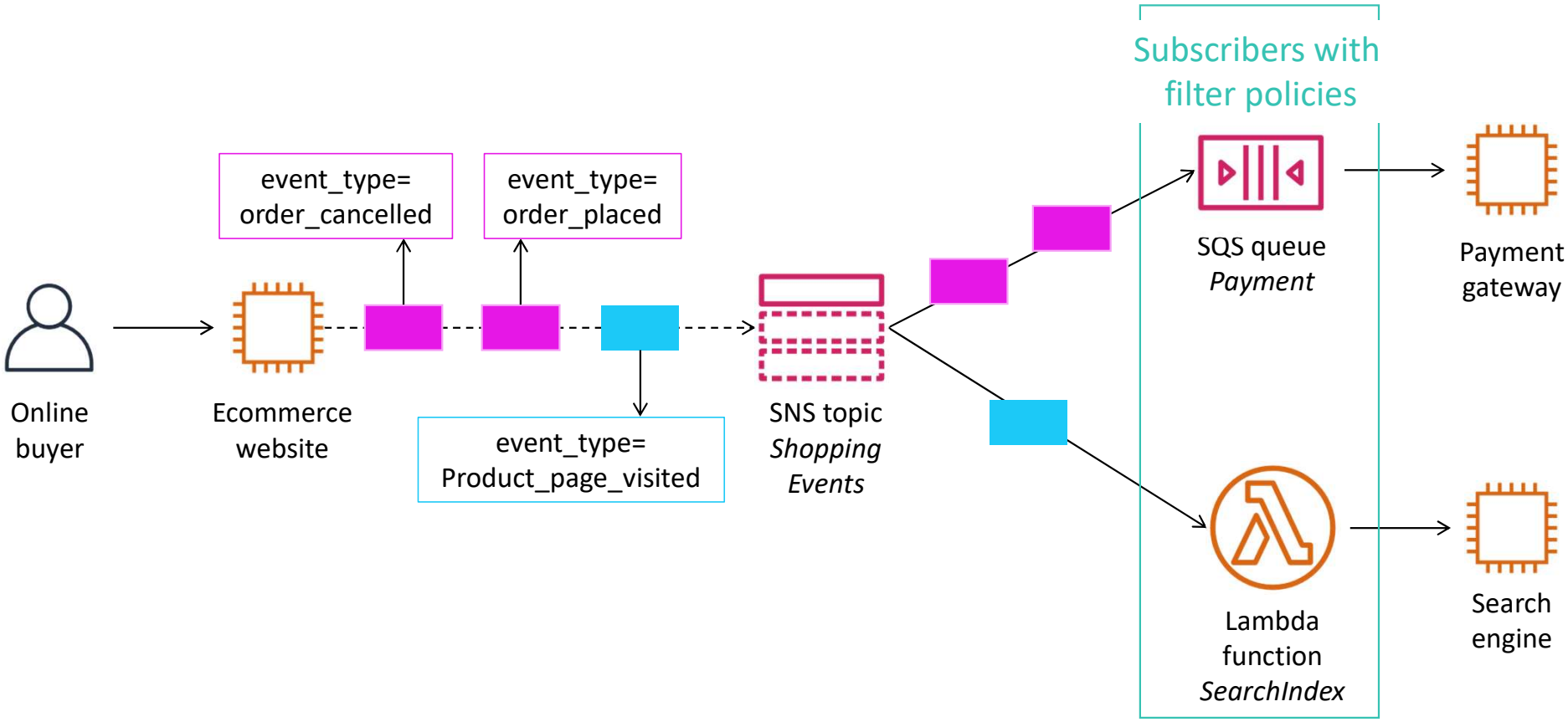
Amazon SNS encodes the message in JSON format and adds metadata.

With raw message delivery enabled, Amazon SNS delivers the message as is.

```
New order!
```

```
{"orderId":10,"orderDate":"2015/10/10","orderDetails":"Thermometer"}
```

Filter policies



Message filtering example

```
topic_arn = sns.create_topic(
        Name='ShoppingEvents'
)['TopicArn']
```

Subscription Topic

```
search_engine_subscription_arn = sns.subscribe(
        TopicArn = topic_arn,
        Protocol = 'lambda',
        Endpoint = 'arn:aws:lambda:us-east-1:123456789012:function:SearchIndex'
)['SubscriptionArn']

sns.set_subscription_attributes(
        SubscriptionArn = search_engine_subscription_arn,
        AttributeName = 'FilterPolicy'
        AttributeValue = '{"event_type": ["product_page_visited"]}'
)
```

Filter Policy

Attribute in filter policy

Message filtering example

```
message = '{"product": {"id": 1251, "status": "in_stock"},'\
          ' "buyer": {"id":4454}}'

sns.publish(
       TopicArn = topic_arn,
       Subject = "Product Visited #1251',
       Message = message,
       MessageAttributes = {
              'event_type': {
                     'DataTye': 'String',
                     'StringValue': 'product_page_visited'
              }
       }
}
```

Message

Message attribute that matches attribute in filter policy

Amazon SNS security

- IAM policies and Amazon SNS policies
- Server-side encryption and AWS KMS
- Amazon VPC