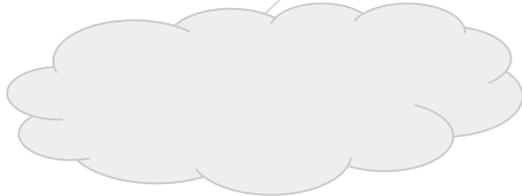


A network diagram consisting of 12 gray circular nodes connected by thin gray lines. The nodes are arranged in a roughly circular pattern, with some lines crossing each other. The text 'AWS Messaging Solutions' is centered over the diagram.

# **AWS Messaging Solutions**



# The Players

## Amazon MQ

Managed implementation of an Apache MQ broker, for legacy migration

## Simple Queue Service (SQS)

Traditional message queue

## Simple Notification Service (SNS)

Pub-sub with a variety of destinations

## Kinesis

Scalable persistent log

---

# Simple Queue Service (SQS)

---

# When You'd Use It

Distribute asynchronous tasks

As-of June 2018, these tasks can be Lambdas

Sequence steps of a distributed workflow

# Features

Any number of producers or consumers

Messages may be up to 256kbytes

Warning: messages are limited to characters allowed by XML 1.0 spec

Each message consumed once, but may be retried

Consumers must explicitly delete messages

Queue defines a “visibility timeout”: after it expires, another consumer may read message

After queue-defined limit of attempts, message is put on dead letter queue

No ordering guarantees from standard queue

FIFO queue introduces ordering guarantees, transaction limits

# Producer

## Standard queue, simplified API:

1. Call `SendMessage` with queue name and message body

## Standard queue, full API:

1. Construct `SendMessageRequest` with queue name and message body
2. Optional: add attributes (metadata)
3. Optional: specify delay before message becomes visible (up to 15 minutes)
4. Call `SendMessage` with request
5. Optional: extract information from response (eg, message ID)

## FIFO queue:

1. Construct `SendMessageRequest` with queue name, group ID, deduplication ID, and message body
2. Optional: specify delay, attributes
3. Call `SendMessage` with request
4. Optional: extract information from response

# Consumer

Short Polling (returns immediately, may not see queued messages):

1. Construct `ReceiveMessageRequest` with queue URL (not name!) and max message count
2. Call `receiveMessage()`
3. For each retrieved message:
  - 3.1. Process message
  - 3.2. Delete message

Long Poll (guaranteed to return message if any in queue):

1. Construct `ReceiveMessageRequest` with queue URL (not name!) and max message count
2. Set `WaitTimeSeconds` request parameter to a non-zero value
3. Call `receiveMessage()`
4. For each retrieved message:
  - 4.1. Process message
  - 4.2. Delete message

---

# **Simple Notification Service (SNS)**

---

# When You'd Use It

Send notifications directly to users

Email, Text Message, Mobile Push Notification

Distribute event notifications to multiple consumers (event bus)

HTTP(S), SQS, Lambda

# Features

Send *same* message to many *subscribed* consumers

Applications must subscribe via intermediary: SQS or HTTP endpoint

SNS/Email/HTTP subscriptions require confirmation from subscriber

Subscribers can define a filter that limits the messages received

Messages may be up to 256 kbytes (140 bytes for SMS)

Messages may have up to 10 “attributes” that provide metadata

# Producer -- Broadcast

## Send same message to all subscribers

1. Create `PublishRequest` with topic ARN (not name!) and message
2. Optional: add subject and message attributes
3. Call `Publish` (returns message ID)

## Send different message depending on protocol

1. Create `PublishRequest` with topic ARN (not name!)
2. Set `MessageStructure` to "json"
3. Set `Message` to JSON blob different *message* values per destination
4. Optional: add subject and message attributes
5. Call `Publish` (returns message ID)

# Producer -- Mobile Push Notifications

## On server (before registering any devices):

1. Call `CreatePlatformApplication` for every service that you want to support
2. Save returned platform ARN

## On device

1. Retrieve token from device-specific service
2. Send token to server (app-specific)

## On server (when client registers):

1. Call `CreatePlatformEndpoint` to register client, using platform ARN
2. Save returned endpoint ARN

## On server (to send notification):

1. Call `Publish` with endpoint ARN and message

# Consumer

## Email / SMS

1. Call `Subscribe`, providing topic ARN (not name!), protocol, and email/phone number
2. Manual: click confirmation link on device

## HTTP

1. Call `Subscribe`, providing topic ARN, protocol (“http”), and destination URL
2. Wait for subscription confirmation request: a POST with JSON body
3. Verify that confirmation request message type is “`SubscriptionConfirmation`”
4. Extract `SubscribeURL` from message body, and perform a GET to that URL
5. Optional: parse response (XML) and save `SubscriptionArn` (to unsubscribe)
6. Listen for notifications: a POST with JSON body and message type “`Notification`”

## SQS

1. Grant topic `SQS:SendMessage` permission on the queue
2. Call `Subscribe`, providing topic ARN, protocol (“sqs”), and queue ARN

---

# Kinesis

---

# When You'd Use It

Distribute events to multiple consumers that may not always be active

Buffer for events, especially if you need to deduplicate

Log aggregation

Streaming data analysis (with Kinesis Analytics)

# Features

Each stream has one or more shards

Each shard can accept 1,000 messages or 1 MB per second

Producer uses “partition key” to assign messages to different shards

Messages are retained up to 7 days (default 1 day)

Consumer reads each shard independently, using “shard iterator”

- Can read from start of stream, end of stream, timestamp or saved sequence number

Each shard may be read 5 times per second with 2 MB/second throughput

- If you have a lot of consumers you may need to create a hierarchy of streams

# Producer

## PutRecord API

1. Construct and send request
2. Retry on `ProvisionedThroughputExceededException`

## PutRecords API

1. Construct request, adding up to 500 records / 5 MB
2. Send request
3. Retry entire request on `ProvisionedThroughputExceededException`
4. Iterate through response records, saving any with throughput-exceeded errors
5. Construct new request with errors from previous, retry

# Consumer

1. Call `DescribeStream` to identify shards
2. For each shard:
  - 2.1. Call `GetShardIterator`
  - 2.2. Call `GetRecords`
  - 2.3. Process records
  - 2.4. Save sequence number of last record processed
3. Sleep, to avoid reading too frequently

---

**For More Information**

---

## SQS Developer Guide

<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html>

## SNS Developer Guide

<https://docs.aws.amazon.com/sns/latest/dg/welcome.html>

## Kinesis Developer Guide

<https://docs.aws.amazon.com/streams/latest/dev/introduction.html>

## Example Code

<https://gist.github.com/kdgregory/84fe6fc26ee6424ecc12926565fee304>