AMAZON SNS

A FULLY-MANAGED MESSAGING SERVICE IN THE CLOUD

INTRODUCTION 👋

Amazon **S**imple **N**otification **S**ervice (**SNS**) is a managed, highly-scalable messaging service by AWS which evolves around **topics**.
If you think about topics, imagine a collection of messages. A client (e.g. a user or an endpoint, like at a HTTP server) can **subscribe** to such a topic and every time a message is published to this topic, **all** clients are notified.

EVENT SOURCES & DESTINATIONS ⚡

Many AWS services are supported as event sources for SNS, including popular ones like **Lambda**, **Step Functions**, **ECS**, **DynamoDB**, **RDS**, **CodeBuild**, **CloudFormation** & **CloudTrail**.

Destinations are grouped into:

- application-to-application (**A2A**) messaging
- application-to-person (**A2P**) notifications

A2A destinations include **Kinesis Data Firehorse**, **Lambda**, **SQS**, **Event Fork Pipelines** and **HTTP/s**.
For A2P you can use **SMS**, **email**, **platform endpoints** (e.g. to deliver events to mobile phones via push notifications), **Chatbot** & **PagerDuty**.

⚠ **Careful with FIFO topics**: Most AWS services only support receiving events from SNS standard topics.

PRICING 💸

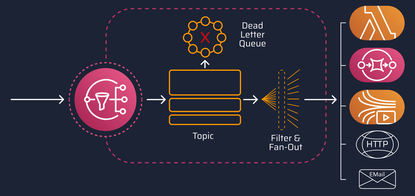
As with other on-demand services, there's no minimum fee. You're paying for actual usage.

- **$0.50** per **1m** SNS requests
- **$0.06** per **100k** notification deliveries via HTTP
- **$2.00** per **100k** deliveries over email

💡 **Free Tier**: covers 1m requests, 100k notifications, 100 SMS & 1000 notifications via email **each month**.

PUBLISH / SUBSCRIBE ⚡

SNS follows the "publish-subscribe"-messaging paradigm, meaning that messages are actively pushed to clients, eliminating the need to poll.



DATA ENCRYPTION 🔒

You can protect message data while

- **in-transit** (sent from & to SNS) and
- **at rest** (while stored at AWS data centers)

For securing your messages in transit, you should use **SSL** and/or **client-side encryption**.
By enabling **S**erver-**S**ide **E**ncryption (**SSE**), SNS will store **sensitive data** (=> the body of a message) in encrypted topics.

What's not encrypted:

- **Topic metadata** - topic names & attributes
- **Message metadata** - subject, message ID, timestamp, and attributes
- **Metrics for topics**

MESSAGE ATTRIBUTES 📝

With SNS, you can provide **structured metadata** which is send together with the message body.

This can be used to provide receivers information about how to process messages, **without processing the message body first**. Message attributes can also be used together with **filter policies** to determine if messages should be delivered to a specific subscriber in the first place.

ACCESS CONTROL 🔑

You can control access to topics via AWS Identity & Access Management (**IAM**) by creating **Topic Policies**.

For example:

- restrict access to **specific users** or **accounts**
- restrict subscribers to use a **specific protocol** like HTTP

```
{
  "Statement": [{
    "Sid": "HttpAccess",
    "Effect": "Allow",
    "Principal": {
      "AWS": "111122223333"
    },
    "Action": ["sns:Subscribe"],
    "Resource": "arn:aws:sns:us-east-1:012345678901:MyTopic",
    "Condition": {
      "StringEquals": {
        "sns:Protocol": "https"
      }
    }
  }]
}
```

ARCHIVING 💾

For archiving messages (e.g. for compliance requirements or to run analytics), you can fan out messages to **Kinesis Data Firehorse**.

This enables you to forward notifications to storage and analytic destinations which are supported by Data Firehorse like **S3** or **Redshift**.

MESSAGE FILTERING

By default, **all** of your topic's subscribers will receive **all messages** that are published.
By assigning **filter policies** to subscriptions, this can be reduced to a subset of messages only relevant for the specific client.

A filter policy defines matching rules that evaluate the **Message Attributes** to determine if a message is **rejected** or **accepted**.

```
{
  "Type": "Notification",
  "MessageId": "87e4eef5-05e0-e2a2-fb90-c160718404 7d",
  "TopicArn": "arn:aws:sns:us-east-2:123456789012:MyTopic",
  "Message": "<message-body>",
  "Timestamp": "2022-06-16T21:12:02.631Z",
  "SignatureVersion": "4",
  "Signature": "signature",
  "UnsubscribeURL": "unsubscribe-url",
  "MessageAttributes": {
    "event": {
      "Type": "String",
      "Value": "payment_received"
    }
  }
}
```

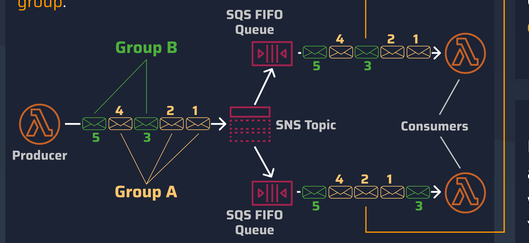
```
{
  "event": ["payment_received"]
}
```

```
{
  "event": [{"anything-but": "payment_received"}]
}
```

MESSAGE ORDERING 🏅

There are a lot of use-cases where it's critical to deliver messages in the order they where published.
To cover this, SNS offers **FIFO** (**F**irst **I**n, **F**irst **O**ut) topics that can be used together with SQS FIFO queues to provide strict message order and delivery.

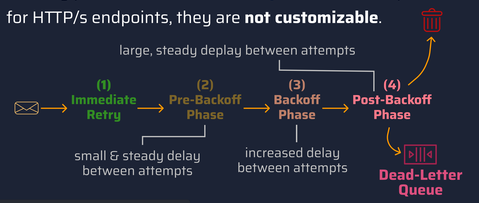
If messages belong to the same group, it's also guaranteed that they are processed in a **strict order** relative to their group.



DELIVERY RETRIES

For each delivery protocol, SNS defines a **delivery policy**.
Those policies define how retries are handled if errors occur.

Delivery policies contain **different phases** and except for HTTP/s endpoints, they are **not customizable**.




```
{
  "healthyRetryPolicy": {
    "minDelayTarget": 1,
    "maxDelayTarget": 60,
    "numRetries": 50,
    "numNoDelayRetries": 3,
    "numMinDelayRetries": 2,
    "numMaxDelayRetries": 35,
    "backoffFunction": "exponential"
  },
  "sicklyRetryPolicy": null,
  "throttlePolicy": {
    "maxReceivesPerSecond": 10
  },
  "guaranteed": false
}
```

A **custom HTTP/s delivery policy** enables you to define how each phase should be handled by SNS.

DEDUPLICATION 👥

Like a reliable message ordering, often you need to make sure messages are only **delivered once to each consumer**.

This can be achived with the same setup as with reliable message ordering (**SNS FIFO topic & SQS FIFO queues**) and messages that contain a unique **deduplication ID**.

Another approach: make sure that the **same message** is delivered once to each of your consumers by enabling **content-based deduplication**.

DEAD-LETTER QUEUES ☠

If messages can't be delivered to clients - for example due to a deleted endpoint or misconfigured IAM policies - you don't want them to just **dissapear** into the Nirvana.
You want to either keep them for analysis purposes or to reprocess them at a later state.

By configuring a SQS queue as your Dead-Letter Queue (DLQ), you can automatically redrive messages after the delivery policy is exhausted.

MONITORING 🌱

CloudWatch offers you a set of predefined metrics for SNS like **published** & **delivered messages**, the **average payload size** or **messages that had been redelivered** to a dead letter queue.

With 3rd-Party service likes Dashbird.io, you can also easily track metrics and get alerted if thresholds are breached. Additionally, you get **well-architected hints** for best practices.

aws