What is the difference between Amazon SNS and Amazon SQS?

Asked 12 years ago Modified 2 months ago Viewed 325k times

Part of AWS Collective

749

When would I use SNS versus SQS, and why are they always coupled together?

amazon-web-services amazon-sqs amazon-sns

Share

Improve this question

Follow

edited Feb 2, 2021 at 10:42

Peter Mortensen

31.6k ● 22 ● 109 ● 133

asked Dec 3, 2012 at 10:22

Nick Ginanto

32.1k ● 49 ● 141 ● 250

1 What I understand from your comment is described in following flow.. is it correct? Publisher --> SNS --> SQL (holding messages in queue) --> Subscriber (currently offline) – friendyogi Oct 28, 2013 at 14:39

5 aws.amazon.com/blogs/aws/… – Manish Jain Mar 29, 2016 at 5:02

4 @friendyogi you mean SQS and not SQL? – elena Apr 17, 2019 at 18:16

wisdomplexus.com/blogs/amazon-sqs-vs-sns – Shiwangini Nov 8, 2020 at 15:25

There are Wikipedia articles for both: *Amazon SNS* and *Amazon SQS* – Peter Mortensen Feb 2, 2021 at 10:40

9 Answers

Sorted by: Highest score (default)

1013

✓

SNS is a distributed **publish-subscribe** system. Messages are **pushed** to subscribers as and when they are sent by publishers to SNS.

SQS is distributed **queuing** system. Messages are *not* pushed to receivers. Receivers have to **poll or pull** messages from **SQS**. Messages can't be received by multiple receivers at the same time. Any one receiver can receive a message, process and delete it. Other receivers do not receive the same message later. Polling inherently introduces some latency in message delivery in SQS unlike SNS where messages are immediately pushed to subscribers. SNS supports several end points such as email, SMS, HTTP end point and SQS. If you want unknown number and type of subscribers to receive messages, you need SNS.

You don't have to couple SNS and SQS always. You can have SNS send messages to email, SMS or HTTP end point apart from SQS. There are advantages to coupling SNS with SQS. You may not want an external service to make connections to your hosts (a firewall may block all incoming connections to your host from outside).

Your end point may just die because of heavy volume of messages. Email and SMS maybe not your choice of processing messages quickly. By coupling SNS with SQS, you can receive messages at your pace. It allows clients to be offline, tolerant to network and host failures. You also achieve guaranteed delivery. If you configure SNS to send messages to an HTTP end point or email or SMS, several failures to send message may result in messages being dropped.

SQS is mainly used to decouple applications or integrate applications. Messages can be stored in SQS for a short duration of time (maximum 14 days). SNS distributes several copies of messages to several subscribers. For example, let’s say you want to replicate data generated by an application to several storage systems. You could use SNS and send this data to multiple subscribers, each replicating the messages it receives to different storage systems (S3, hard disk on your host, database, etc.).

Share Improve this answer Follow

edited Feb 2, 2021 at 9:50
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Dec 3, 2012 at 22:14
Srikanth
12k ● 3 ● 26 ● 28

---

3 so basically, to implement something like push notification messages, it is recommended to use SNS and SQS so the pushes with sns will be queued until the user is only to

retrieve them from the queue? Is it possible to create a queue per user? – Nick Ginanto Dec 4, 2012 at 5:20

3 Yeah. You can have as many subscribers as you want for SNS. You can have notifications sent to multiple queues. – Srikanth Dec 4, 2012 at 20:14

5 @NickGinanto Queue per user probably isn't what you want. You'd probably want one queue for each service that then handles user-specific messages. This diagram may help: aws.amazon.com/blogs/aws/… – Trenton Jul 28, 2015 at 16:27

4 It should probably be noted that as of mid 2018 SQS can trigger lambdas and therefore is more akin to a pubsub in that case. – cyberwombat Jul 19, 2019 at 2:37

3 This is by far, the best and simple explanation I came across about the two AWS services. – Yajairo87 Aug 14, 2019 at 5:31

425

Here's a comparison of the two:

Entity Type

- SQS: Queue (Similar to JMS)
- SNS: Topic (Pub/Sub system)

Message consumption

- SQS: Pull Mechanism - Consumers poll and pull messages from SQS
- SNS: Push Mechanism - SNS Pushes messages to consumers

Use Case

- SQS: Decoupling two applications and allowing parallel asynchronous processing
- SNS: Fanout - Processing the same message in multiple ways

Persistence

- SQS: Messages are persisted for some (configurable) duration if no consumer is available (maximum two weeks), so the consumer does not have to be up when messages are added to queue.
- SNS: No persistence. Whichever consumer is present at the time of message arrival gets the message and the message is deleted. If no consumers are available then the message is lost after a few retries.

Consumer Type

- SQS: All the consumers are typically identical and hence process the messages in the exact same way (each message is processed once by one consumer, though in rare cases messages may be resent)
- SNS: The consumers might process the messages in different ways

Sample applications

- SQS: Jobs framework: The Jobs are submitted to SQS and the consumers at the other end can process the jobs asynchronously. If the job frequency increases, the number of consumers can simply be increased to achieve better throughput.
- SNS: Image processing. If someone uploads an image to S3 then watermark that image, create a thumbnail and also send a *Thank You* email. In that case S3 can publish notifications to an SNS topic with three consumers listening to it. The first one watermarks the image, the second one creates a thumbnail and the third one sends a *Thank You* email. All of them receive the same message (image URL) and do their processing in parallel.

Share Improve this answer Follow

edited Feb 2, 2021 at 10:24
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Jun 14, 2018 at 6:49
Arafat Nalkhande
11.7k ● 9 ● 44 ● 66

3 if no consumers are available then there is a retry mechanism, even default value is 10 retries. – Arpit Solanki May 7, 2019 at 11:10

Nice detailed post. We have different messages for different consumers - What should we do - use SNS and define different topics or use SQS and define different queues? A topic/queue could though have one or more consumers. – Andy Dufresne May 9, 2019 at 14:07

2 I do not think "SQS : All the consumers are supposed to be identical and hence process the messages in exact same way" this is right. I have used SQS where two different AWS services are picking up from SQS queue and processing the message in their own way (different application logic in those different services). Am I missing something? – nad Mar 25, 2020 at 16:49

1 @nad I will need to understand your use case but to me it makes no sense that 2 SQS consumers are processing messages in non-identical ways. That is a use case for SNS – Arafat Nalkhande Mar 27, 2020 at 10:29

1 Yes, but then who deletes the message. Wouldnt the consumer 1 and connsumer 2 and consumer 3 all keep getting the same message again and again – Arafat Nalkhande Jul 10, 2020 at 17:08

71

You can see **SNS** as a traditional topic which you can have multiple Subscribers. You can have heterogeneous subscribers for one given SNS topic, including Lambda and SQS, for example. You can also send SMS messages or even e-mails out of the box using SNS. One thing to consider in SNS is only one message (notification) is received at once, so you cannot take advantage from batching.

SQS, on the other hand, is nothing but a *queue*, where you store messages and subscribe one consumer (yes, you can have N consumers to one SQS queue, but it would get messy very quickly and way harder to manage considering all consumers would need to read the message at least once, so one is better off with SNS combined with SQS for this use case, where SNS would push notifications to N SQS queues and every queue would have one subscriber, only) to process these messages. As of Jun 28, 2018, AWS Supports Lambda Triggers for SQS, meaning you don't have to **poll** for messages any more.

Furthermore, you can configure a DLQ on your source SQS queue to send messages to in case of failure. In case of success, messages are automatically deleted (this is another great improvement), so you don't have to worry about the already processed messages being read again in case you forgot to delete them manually. I suggest taking a look at Lambda Retry Behaviour to better understand how it works.

One great benefit of using SQS is that it enables batch processing. Each batch can contain up to 10 messages, so if 100 messages arrive at once in your SQS queue, then 10 Lambda functions will spin up (considering the default auto-scaling behaviour for Lambda) and they'll process these 100 messages (keep in mind this is the happy path as in practice, a few more Lambda functions could spin up reading less than the 10 messages in the batch, but you get the idea). If you posted these same 100 messages to SNS, however, 100 Lambda functions would spin up, unnecessarily increasing costs and using up your Lambda concurrency.

However, if you are still running traditional servers (like EC2 instances), you will still need to poll for messages and manage them manually.

You also have **FIFO SQS queues**, which guarantee the delivery order of the messages. SQS FIFO is also supported as an event source for Lambda as of November 2019

Even though there's some overlap in their use cases, both SQS and SNS have their own spotlight.

Use **SNS** if:

- multiple subscribers is a requirement
- sending SMS/E-mail out of the box is handy

Use **SQS** if:

- only one subscriber is needed
- batching is important

Share Improve this answer
Follow

edited Feb 2, 2021 at 10:37
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Mar 20, 2019 at 13:24
Thales Minussi
7,215 ● 1 ● 32 ● 50

Distributing tasks among *many* consumers is a primary intended use case for SQS. (Regardless of whether consumers retrieve 1 message or 10 at a time, it is up to the application to control how many consumers exist concurrently.) It differs from SNS because SNS tries to send every message to all subscribers, whereas SQS tries to partition the messages among the consumers. – benjimin Dec 19, 2021 at 23:28

Max batch size is now 10k for Standard Queues; still 10 for FIFO Queues. – rinogo Jan 29 at 15:55

67

AWS SNS is a publisher subscriber network, where subscribers can subscribe to topics and will receive messages whenever a publisher publishes to that topic.

AWS SQS is a queue service, which stores messages in a queue. SQS cannot deliver any messages, where an external service (lambda, EC2, etc.) is needed to poll SQS and grab messages from SQS.

SNS and SQS can be used together for multiple reasons.

1. There may be different kinds of subscribers where some need the immediate delivery of messages, where some would require the message to persist, for later usage via polling. See this link.
2. The "**Fanout Pattern**." This is for the asynchronous processing of messages. When a message is published to SNS, it can distribute it to multiple SQS queues in parallel. This can be great when loading thumbnails in an application in parallel, when images are being published. See this link.
3. **Persistent storage**. When a service that is going to process a message is not reliable. In a case like this, if SNS pushes a notification to a Service, and that

service is unavailable, then the notification will be lost. Therefore we can use SQS as a persistent storage and then process it afterwards.

Share Improve this answer

Follow

edited Feb 2, 2021 at 10:14

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Apr 18, 2018 at 5:36

Keet Sugathadasa
13.4k ● 9 ● 76 ● 84

---

8 Your comment is the only one clearly explaining how SNS and SQS can be combined. Thanks! – Andriy Oct 19, 2020 at 11:06

I am a newbie to AWS, but has 3. changed since the answer was written? [docs.aws.amazon.com/sns/latest/dg/… Delivery Retries) *Amazon SNS defines a delivery policy for each delivery protocol. The delivery policy defines how Amazon 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, Amazon SNS stops retrying the delivery and discards the message—unless a dead-letter queue is attached to the subscription.* – buzztr Mar 22, 2021 at 9:05

@buzztr 3. hasn't changed; it refers to the consumer of the SQS becoming unavailable. That documentation you cite describes a new feature of SNS in which failure to deliver a message to the consumer(s) (in this case, the SQS) after exhausting your retries results in SNS putting that message put onto a dedicated SQS queue for to handle transmission error. This means instead of the message being lost entirely, you have it saved and can re-attempt transmission through the SNS topic at a later date. – Hans Hermans May 24, 2021 at 1:37

37

From the AWS documentation:

> Amazon SNS allows applications to send time-critical messages to multiple subscribers through a "push" mechanism, eliminating the need to periodically check or "poll" for updates.
>
> Amazon SQS is a message queue service used by distributed applications to exchange messages through a polling model, and can be used to decouple sending and receiving components—without requiring each component to be concurrently available.

Fanout to Amazon SQS queues

Share Improve this answer Follow

edited Feb 2, 2021 at 9:52

Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered Mar 12, 2014 at 4:31

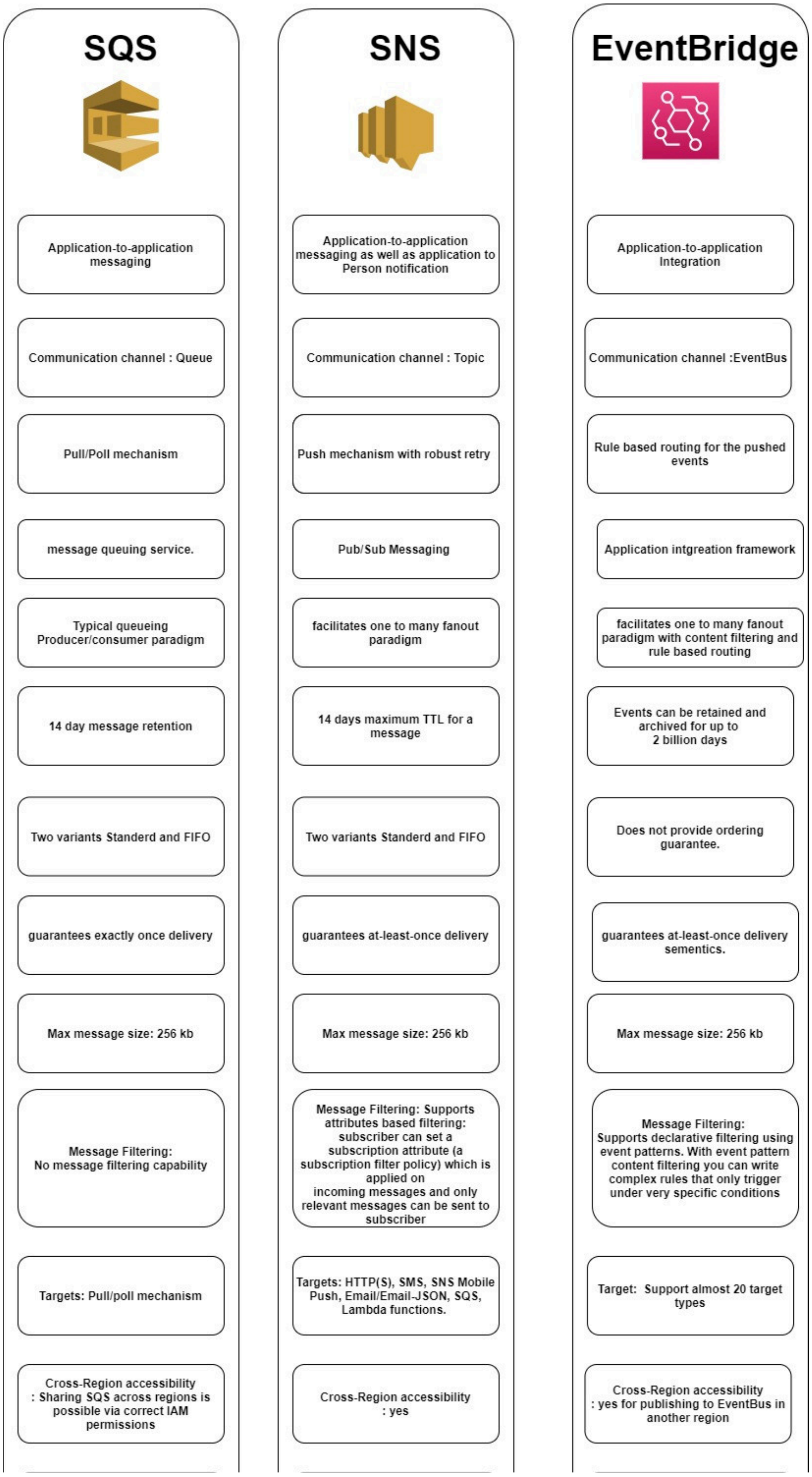
Tommy
8,138 ● 4 ● 32 ● 26

1 SQS can "poll waiting for a message to come in" (which is similar to a push). But it only processes that message once so it's not a broadcast push, FWIW :) – rogerdpack Aug 12, 2020 at 16:52

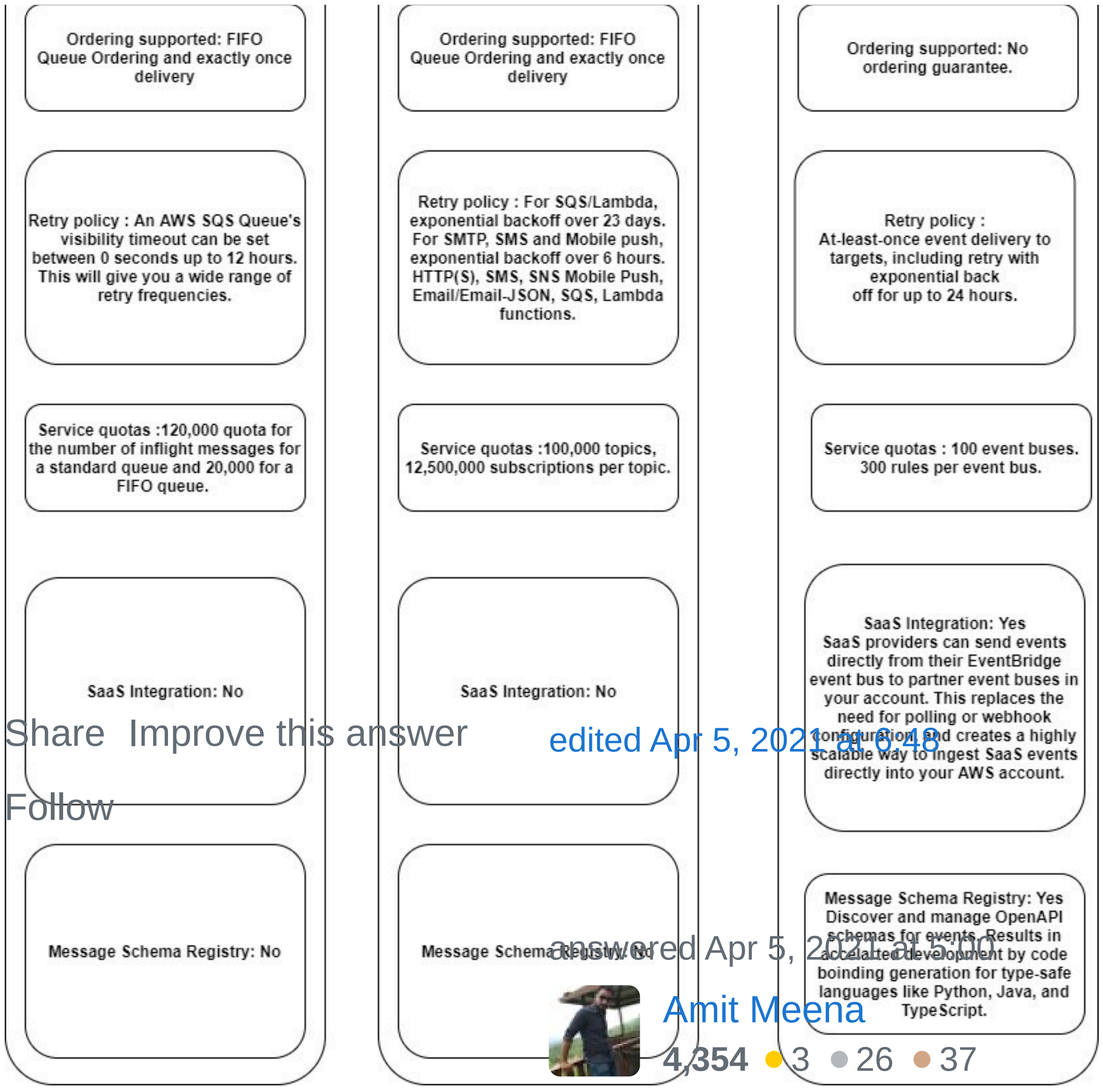
23

Following are the major differences between the main messaging technologies on AWS (SQS, SNS, +EventBridge). In order to choose a particular AWS service, we should know the functionalities a service provides as well as its comparison with other services.

The below diagram summarizes the main similarities as well as differences between this service.



SQS	SNS	EventBridge
Application-to-application messaging	Application-to-application messaging as well as application to Person notification	Application-to-application Integration
Communication channel : Queue	Communication channel : Topic	Communication channel :EventBus
Pull/Poll mechanism	Push mechanism with robust retry	Rule based routing for the pushed events
message queuing service.	Pub/Sub Messaging	Application intgreation framework
Typical queueing Producer/consumer paradigm	facilitates one to many fanout paradigm	facilitates one to many fanout paradigm with content filtering and rule based routing
14 day message retention	14 days maximum TTL for a message	Events can be retained and archived for up to 2 billion days
Two variants Standerd and FIFO	Two variants Standerd and FIFO	Does not provide ordering guarantee.
guarantees exactly once delivery	guarantees at-least-once delivery	guarantees at-least-once delivery sementics.
Max message size: 256 kb	Max message size: 256 kb	Max message size: 256 kb
Message Filtering: No message filtering capability	Message Filtering: Supports attributes based filtering: subscriber can set a subscription attribute (a subscription filter policy) which is applied on incoming messages and only relevant messages can be sent to subscriber	Message Filtering: Supports declarative filtering using event patterns. With event pattern content filtering you can write complex rules that only trigger under very specific conditions
Targets: Pull/poll mechanism	Targets: HTTP(S), SMS, SNS Mobile Push, Email/Email-JSON, SQS, Lambda functions.	Target: Support almost 20 target types
Cross-Region accessibility : Sharing SQS across regions is possible via correct IAM permissions	Cross-Region accessibility : yes	Cross-Region accessibility : yes for publishing to EventBus in another region


Ordering supported: FIFO Queue Ordering and exactly once delivery

Retry policy : An AWS SQS Queue's visibility timeout can be set between 0 seconds up to 12 hours. This will give you a wide range of retry frequencies.

Service quotas :120,000 quota for the number of inflight messages for a standard queue and 20,000 for a FIFO queue.

SaaS Integration: No

Message Schema Registry: No

Ordering supported: FIFO Queue Ordering and exactly once delivery

Retry policy : For SQS/Lambda, exponential backoff over 23 days. For SMTP, SMS and Mobile push, exponential backoff over 6 hours. HTTP(S), SMS, SNS Mobile Push, Email/Email-JSON, SQS, Lambda functions.

Service quotas :100,000 topics, 12,500,000 subscriptions per topic.

SaaS Integration: No

Message Schema Registry: No

Ordering supported: No ordering guarantee.

Retry policy : At-least-once event delivery to targets, including retry with exponential back off for up to 24 hours.

Service quotas : 100 event buses. 300 rules per event bus.

SaaS Integration: Yes SaaS providers can send events directly from their EventBridge event bus to partner event buses in your account. This replaces the need for polling or webhook configuration and creates a highly scalable way to ingest SaaS events directly into your AWS account.

Message Schema Registry: Yes Discover and manage OpenAPI schemas for events. Results in accelerated development by code boinding generation for type-safe languages like Python, Java, and TypeScript.

Share Improve this answer Follow

edited Apr 5, 2021 at 6:48

answered Apr 5, 2021 at 5:00

Amit Meena

4,354 ●3 ●26 ●37

13

In simple terms,

- SNS - sends messages to the subscriber using push mechanism and no need of pull.
- SQS - it is a message queue service used by distributed applications to exchange messages

> through a polling model, and can be used to decouple sending and receiving components.

A common pattern is to use SNS to publish messages to Amazon SQS queues to reliably send messages to one or many system components asynchronously.

Reference from *Amazon SNS FAQs*.

Share Improve this answer Follow

edited Feb 2, 2021 at 10:38
Peter Mortensen
31.6k ● 22 ● 109 ● 133

answered May 27, 2019 at 20:13
Krunal Barot
932 ● 6 ● 17

1 SQS cannot send a message to ***many systems*** as it does not fan out the messages. Yes, many pollers can pull messages from it, but if one the consumers deletes the message then other subscribers won't be able to consume the same message again. SNS is preferred over SQS if you want to achieve a fan out pattern. Also, if the `visibilityTimeout` is set, than no other systems will be able to consume the message once it's being processed by other system. – Thales Minussi Jun 7, 2019 at 8:09

2 A common pattern is to use SNS to publish messages to Amazon SQS queues to reliably send messages to one or many system components asynchronously. Reference from aws.amazon.com/sns/faqs – Krunal Barot Jun 7, 2019 at 11:59

1 If that's what you meant (SNS -> multiple SQS queues) then please edit your answer and I will happily remove my

downvote. The way you put it, it sounds like SQS can fan out. – Thales Minussi Jun 7, 2019 at 12:52

1 Yes.. thats where the confusion was. I have edited it .. Thanks :) – Krunal Barot Jun 7, 2019 at 14:31

This is a start, but the question was about the differences, etc.: *"When would I use SNS versus SQS, and why are they always coupled together?"* Perhaps address that more directly? You can edit your answer (***without*** "Edit:", "Update:", or similar - the answer should appear as if it was written today). – Peter Mortensen Feb 2, 2021 at 10:41

7

There are some key distinctions between SNS and SQS:

- **SNS** supports application-to-application **(A2A)** and application-to-person **(A2P)** communication, while SQS supports only **A2A** communication.
- **SNS** is a pub/sub system, while **SQS** is a queuing system. You'd typically use **SNS** to send the same message to multiple consumers via topics. In comparison, in most scenarios, each message in an **SQS** queue is processed by only one consumer. With **SQS**, messages are delivered through a long ***polling*** (pull) mechanism, while **SNS** uses a ***push*** mechanism to immediately deliver messages to subscribed endpoints.
- **SNS** is typically used for applications that need real time notifications, while **SQS** is more suited for message processing use cases.

- **SNS** does not persist messages - it delivers them to subscribers that are present, and then deletes them. In comparison, **SQS** can persist messages (from 1 minute to 14 days).

Individually, **Amazon SQS** and **SNS** are used for different use cases. You can, however, use them together in some scenarios.

Share Improve this answer

Follow

edited Oct 7 at 16:05
Govinda Sakhare
5,709 ● 6 ● 39 ● 81

answered Dec 22, 2022 at 15:07
Viktor
626 ● 9 ● 16

1

One reason for coupling SQS and SNS would be for data processing pipelines.

Let's say you are generating three kinds of product, and that products B & C are both derived from the same intermediate product A. For each kind of product (i.e., for each segment of the pipeline) you set up:

- a compute resource (maybe a lambda function, or a cluster of virtual machines, or an autoscaling kubernetes job) to generate the product.
- a queue (describing units of work that need to be performed) to partition the work across the compute resource (so that each unit of work is processed

exactly once, but separate units of work can be processed separately in parallel and asynchronously with each other).

- a news feed (announcing outputs that have been produced).

Then arrange so that the input queues for B & C are both subscribing to the output announcements of A.

This makes the pipeline modular on the level of infrastructure. Rather than having a monolithic server application that generates all three products together, different stages of the pipeline can utilise different hardware resources (for example, perhaps stage B is very memory intensive, but the two other stages can be performed with cheaper hardware/services). This also makes it easier to iterate on the development of one pipeline segment without disrupting delivery of the other products.

Share Improve this answer Follow

answered Dec 22, 2021 at 9:36

benjimin
4,810 ● 1 ● 37 ● 56