## Functionality

* Unlimited queues and messages: Create unlimited Amazon SQS queues with an unlimited number of message in any region
* Payload Size: Message payloads can contain up to 256KB of text in any format. Each 64KB ‘chunk’ of payload is billed as 1 request. For example, a single API call with a 256KB payload will be billed as four requests. To send messages larger than 256KB, you can use the [Amazon SQS Extended Client Library for Java](https://github.com/awslabs/amazon-sqs-java-extended-client-lib), which uses Amazon S3 to store the message payload. A reference to the message payload is sent using SQS.
* Batches: Send, receive, or delete messages in batches of up to 10 messages or 256KB. Batches cost the same amount as single messages, meaning SQS can be even more cost effective for customers that use batching.
* [Long polling:](http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-long-polling.html) Reduce extraneous polling to minimize cost while receiving new messages as quickly as possible. When your queue is empty, long-poll requests wait up to 20 seconds for the next message to arrive. Long poll requests cost the same amount as regular requests.
* Retain messages in queues for up to 14 days.
* Send and read messages simultaneously.
* Message locking: When a message is received, it becomes “locked” while being processed. This keeps other computers from processing the message simultaneously. If the message processing fails, the lock will expire and the message will be available again.
* Queue sharing: Securely share Amazon SQS queues anonymously or with specific AWS accounts. Queue sharing can also be restricted by IP address and time-of-day.
* **Server-side encryption (SSE):** Protect the contents of messages in Amazon SQS queues using keys managed in the AWS Key Management Service (AWS KMS). SSE encrypts messages as soon as Amazon SQS receives them. The messages are stored in encrypted form and Amazon SQS decrypts messages only when they are sent to an authorized consumer.
* Dead Letter Queues (DLQ): Handle messages that have not been successfully processed by a consumer with Dead Letter Queues. When the maximum receive count is exceeded for a message it will be moved to the DLQ associated with the original queue. Set up separate consumer processes for DLQs which can help analyze and understand why messages are getting stuck. DLQs must be of the same type as the source queue (standard or FIFO).

## Using Amazon SQS with other AWS infrastructure web services

Amazon SQS message queuing can be used with other AWS Services such **as**[**Redshift**](https://aws.amazon.com/redshift/)**,**[**DynamoDB**](https://aws.amazon.com/dynamodb/)**,**[**RDS**](https://aws.amazon.com/rds/)**,**[**EC2**](https://aws.amazon.com/ec2-sla/)**,**[**ECS**](https://aws.amazon.com/ecs/)**,**[**Lambda**](https://aws.amazon.com/lambda/)**, and**[**S3**](https://aws.amazon.com/s3/)**,** to make distributed applications more scalable and reliable. Below are some common design patterns:

* Work Queues: Decouple components of a distributed application that may not all process the same amount of work simultaneously.
* Buffer and Batch Operations: Add scalability and reliability to your architecture, and smooth out temporary volume spikes without losing messages or increasing latency.
* Request Offloading: Move slow operations off of interactive request paths by enqueing the request.
* **Fanout:**[Combine SQS with Simple Notification Service (SNS)](http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqssubscribe.html) to send identical copies of a message to multiple queues in parallel.
* **Priority: Use separate queues to provide prioritization of work.**
* **Scalability:** Because message queues decouple your processes, it’s easy to scale up the send or receive rate of messages - simply add another process.
* **Resiliency:** When part of your system fails, it doesn’t need to take the entire system down. Message queues decouple components of your system, so if a process that is reading messages from the queue fails, messages can still be added to the queue to be processed when the system recovers.

**From:** Vernekar, Shantaram [CCC-OT NE]   
**Sent:** Sunday, July 8, 2018 12:53 PM  
**To:** 'Shantaram Vernekar ([shantaram\_vernekar@persistent.com](mailto:shantaram_vernekar@persistent.com))' <[shantaram\_vernekar@persistent.com](mailto:shantaram_vernekar@persistent.com)>  
**Subject:** AWS - Messaging

**Amazon Simple Queue Service (Amazon SQS)**

Two types of queues – Standard and FIFO

**Standard Queue**

* managed message queuing service
* makes it simple and cost effective to decouple the components of a cloud application.
* a buffer between the application components
* ensures delivery of each message at least once
* supports multiple readers and writers interacting with the same queue
* you should design your system to be idempotent (that is, it must not be adversely affected if it processes the same message more than once).
* the service does not guarantee First In, First Out (FIFO) delivery of messages
* If your system requires that order be preserved, you can place sequencing information in each message so that you can reorder the messages when they are retrieved from the queue.
* Consumer should process the message and delete the message from queue within **visibility timeout period,** otherwise message is added back to queue for processing (message in unhide)
  + During the visibility timeout period message is hidden – no consumer can process it
  + visibility timeout period – starts once the consumer starts processing it.

**FIFO Queue**

High throughput – 300 messages per second (send, receive, or delete operations) .. if 10 per batch than 3000 messages..

The order in which messages are sent and received is strictly preserved.

A message is delivered once and remains available until a consumer processes and deletes it. Duplicates are not introduced into the queue

When to use ? --- Send data between applications when the order of events is important,

When to use Standard queue - Send data between applications when the throughput is important. designed to be massively scalable using a highly distributed architecture

Delay Queues

allow you to postpone the delivery of new messages in a queue for a specific number of seconds. If you create a delay queue, any message that you send to that queue will be invisible to consumers for the duration of the delay period

You can have up to 120,000 messages in flight at any given time. Amazon SQS supports up to 12 hours’ maximum visibility timeout

accessed through HTTP request-response

Queue Operations – Create, delete, list, etc..

Unique IDs - Your messages are identified via a globally unique ID that Amazon SQS returns when the message is delivered to the queue.

When you receive a message from the queue, the response includes a **receipt handle**, which you must provide when deleting the message.

**Queue and Message Identifiers**

three identifiers

* queue URLs  - unique for a given account, contains queue name + other information
* Message IDS – when a message is added, unique message iD is returned. Max message length - 100 characters.
* Receipt Handlers - delete a message, you need the message’s receipt handle instead of the message ID

**Message Attributes**

Message attributes allow you to provide structured metadata items (such as timestamps, geospatial data, signatures, and identifiers) about the message

sent along with, the message body

receiver of the message can use this information to help decide how to handle the message without having to process the message body first

Max - 10 attribute

**Long Polling**

If your SQS client is just a loop that repeatedly checks for new messages, however, then this pattern becomes problematic, as the constant calls to ReceiveMessage burn CPU cycles and tie up a thread

you send a WaitTimeSeconds argument to ReceiveMessage of up to 20 seconds. Call will wait for x seconds for message to appear, if not present. If present it will return right away

Long polling drastically reduces the amount of load on your client.

**Dead letter queues**

Queue to which other queues send messages if not processed successfully

ability to sideline and isolate the unsuccessfully processed messages. You can then analyze any messages sent to the dead letter queue to try to determine the cause of failure

**Access control**

IAM Policies can be granted to queue. Example queue A can send messages to queue B owned by another account

**Tradeoff Message Durability and Latency**

Amazon SQS does not return success to a SendMessage API call until the message is durably stored in Amazon SQS – **Synchronous call**

Default/Max

Visibility timeout – 30 seconds/12 hours

Retention period – 4 days/14 days

Receive message wait time (Longpolling) – 0/20 seconds

Delivery delay(delay queues) – 0/15 seconds

Message size – max 256 kb

Notes:

* Use SQS and SNS- for new brand applications

Amazon MQ – moving an existing application in cloud without code change. MQ supports industry standard APIS and protocols – switch from any standard message broker to MQ.

Amazon MQ is a managed message broker service for Apache ActiveMQ

* SQS v/s SNS 

SQS – pull, decouple components, used by distributed applications to exchange messages..

SNS – push mechanism, send time critical messages to multiple subscribers

* SQS bounded by region – that means a message on a queue in region A cannot be shared with another queue in Region B

<https://aws.amazon.com/sqs/faqs/#fifo-queues>

**Amazon Simple Workflow Service (Amazon SWF)**

coordinate work across distributed components.

a task -  represents a logical unit of work that is performed by a component of your application

stores tasks, assigns them to workers when they are ready, monitors their progress, and maintains their state, including details on their completion

**Workflows** coordinate and manage the execution of activities that can be run asynchronously across multiple computing devices and that can feature both sequential and parallel processing.

Tasks  activities

**Workflow Domains** – workflow from one domain cannot interact with workflow from another domain

* Workflows (one or more)
  + Tasks/Activities (one or more)

**Actors**

Workflow starter  application that can initiate workflow executions (example : Amazon via online application or mobile app)

Decider  logic that coordinates the tasks in a workflow. The decider schedules the activity tasks and provides input data to the activity workers

                                processes events that arrive while the workflow is in progress and closes the workflow when the objective has been completed.

activity worker  single computer process (or thread) that performs the activity tasks in your workflow.

**Tasks**

activity tasks  tells an activity worker to perform its function. Example Inventory/Charge a credit card

AWS Lambda tasks  similar to an activity task, but executes an AWS Lambda function instead of a traditional Amazon SWF activity

decision tasks  tells a decider that the state of the workflow execution has changed so that the decider can determine the next activity that needs to be performed. The decision task contains the current workflow history.

**Task List**

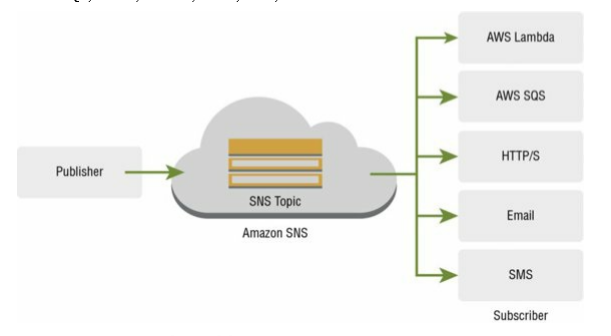
Task lists provide a way of organizing the various tasks associated with a workflow. You could think of task lists as similar to dynamic queues. When a task is scheduled in Amazon SWF, you can specify a queue (task list) to put it in

**Amazon Simple Notification Service (Amazon SNS)**

* notifications being delivered to clients using a push mechanism that eliminates the need to check periodically (or poll) for new information and updates
* In China, you can send messages to Android devices with Baidu Cloud Push. You can use Amazon SNS to send Short Message Service (SMS) messages to mobile device users in the United States or to email recipients worldwide.
* consists of two types of clients: publishers and subscribers

Publishers communicate to subscribers asynchronously by sending a message to a topic. A topic is simply a logical access point/communication channel that contains a list of subscribers and the methods used to communicate to them. When you send a message to a topic, it is automatically forwarded to each subscriber of that topic using the communication method configured for that subscriber.

                                Publisher  topic <-- Subscriber



When using Amazon SNS, you (as the owner)

* create a topic and control access to it by
* defining policies that determine which publishers and subscribers can communicate with the topic and via which technologies.
* Publishers send messages to topics that they created or that they have permission to publish to

**USecases**

provides significant advantages to developers who build applications that rely on real-time events

Event updates and notifications concerning validation, approval, inventory changes, and shipment status are immediately delivered to relevant system components and end users

application that generates or consumes notification

fanout

Example:

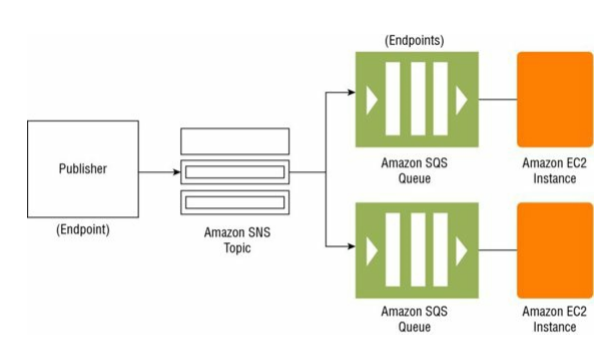
Amazon Order

Produce message to topic

Subscriber 1 – SQS queue  EC2  process/fulfillment of order

Subscriber 2 – SQS queue  . EC2   send order data to a data warehouse application/service for analysis

Load data from prod to DEV/UAT DB



Application and System Alerts

Example : AWS notification  receive immediate notification when an event occurs, such as a specific change to your Auto Scaling group in AWS.

Push Email and Text Messaging

Example : subscribe to newsletters

Mobile Push Notifications

Review Questions

1 – D

2 – A

3 – ACD

4 – A

5 – D

6 – BD  
7 – B

8 – D

9 – B  -- ABC

10 – D

11 – A

12 – D

13 –C - B

14 – D

15 – A - B

16 –D  
17 – B

18 – D - C

19 – C

20 C

SQS:

First service from AMAZON

Example : Travel website

Pull mechanism

Build Autoscaling – brings elasticity to applications

