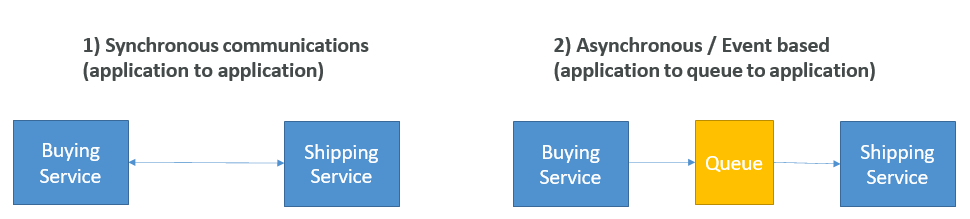
Section Introduction

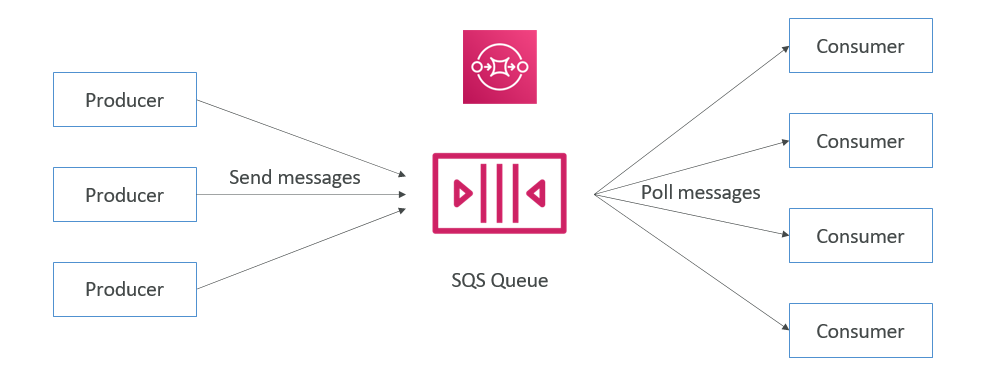
* When we start deploying multiple applications, they will inevitably need to communicate with one another
* There are two patterns of application communication



**Section Introduction**

* Synchronous between applications can be problematic if there are sudden spikes of traffic
* What if you need to suddenly encode 1000 videos but usually it’s 10?
* In that case, it’s better to decouple your applications,
  + using SQS: queue model
  + using SNS: pub/sub model
  + using Kinesis: real-time streaming model
* These services can scale independently from our application!

**Amazon SQS What’s a queue?**

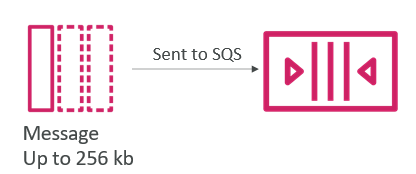


**Amazon SQS – Standard Queue**

* Oldest offering (over 10 years old)
* Fully managed service, used to decouple applications
* Attributes:
  + Unlimited throughput, unlimited number of messages in queue
  + Default retention of messages: 4 days, maximum of 14 days
  + Low latency (<10 ms on publish and receive)
  + Limitation of 256KB per message sent
* Can have duplicate messages (at least once delivery, occasionally)
* Can have out of order messages (best effort ordering)

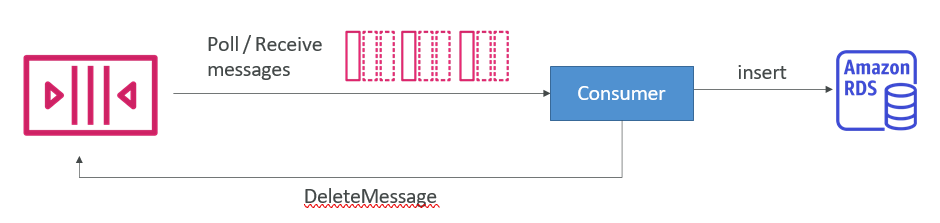
**SQS – Producing Messages**

* Produced to SQS using the SDK (SendMessage API)
* The message is persisted in SQS until a consumer deletes it
* Message retention: default 4 days, up to 14 days
* Example: send an order to be processed
  + Order id
  + Customer id
  + Any attributes you want
* SQS standard: unlimited throughput



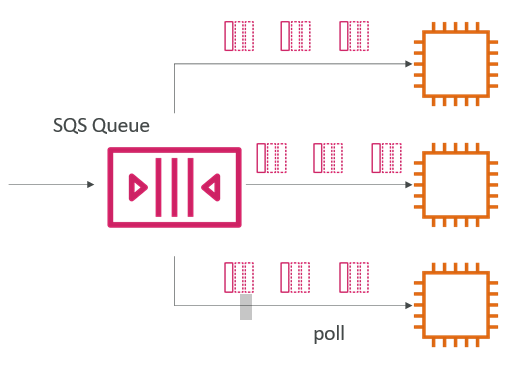
**SQS – Consuming Messages**

* Consumers (running on EC2 instances, servers, or AWS Lambda)…
* Poll SQS for messages (receive up to 10 messages at a time)
* Process the messages (example: insert the message into an RDS database)
* Delete the messages using the DeleteMessage API

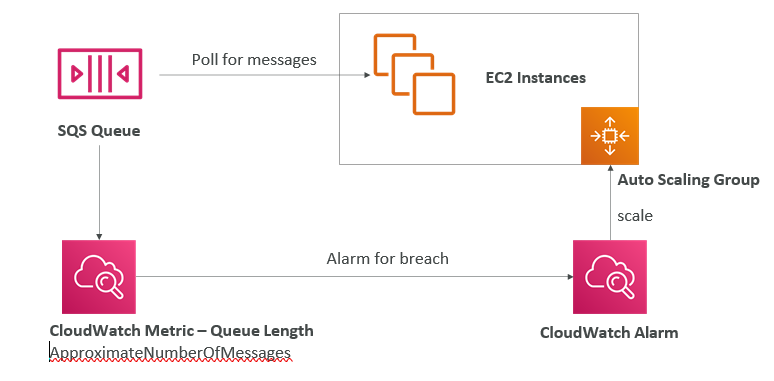


**SQS – Multiple EC2 Instances Consumers**

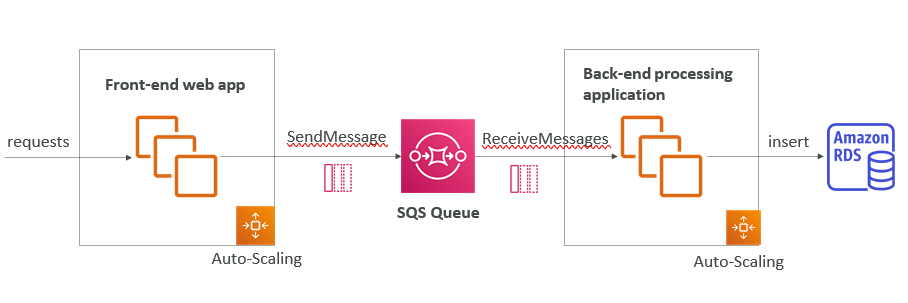
* Consumers receive and process messages in parallel
* At least once delivery
* Best-effort message ordering
* Consumers delete messages after processing them
* We can scale consumers horizontally to improve throughput of processing



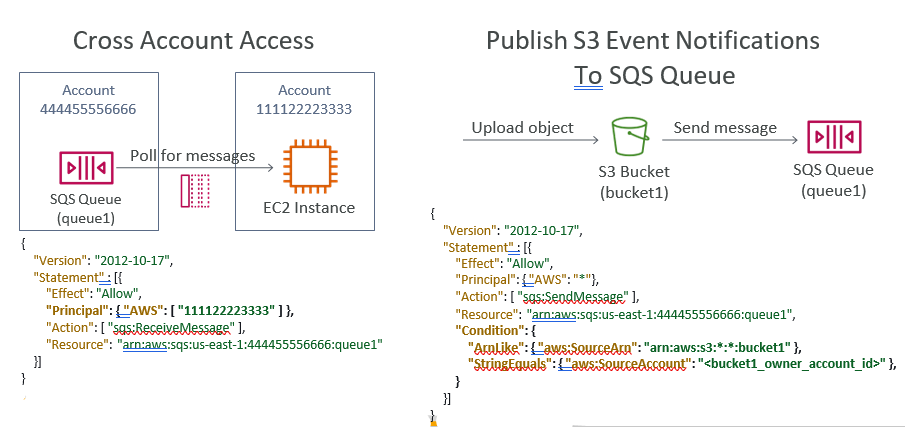
**SQS with Auto Scaling Group (ASG)**



**SQS to decouple between application tiers**

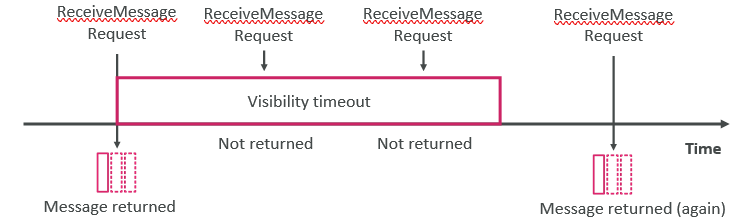


**SQS Queue Access Policy**



**SQS – Message Visibility Timeout**

* After a message is polled by a consumer, it becomes invisible to other consumers
* By default, the “message visibility timeout” is 30 seconds
* That means the message has 30 seconds to be processed
* After the message visibility timeout is over, the message is “visible” in SQS



• If a message is not processed within the visibility timeout, it will be processed twice

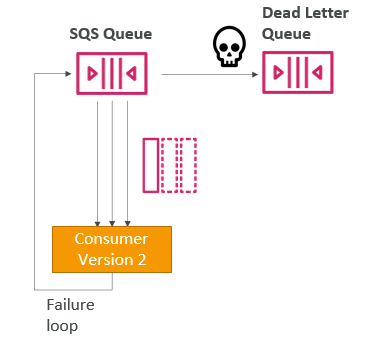
• A consumer could call the ChangeMessageVisibility API to get more time

• If visibility timeout is high (hours), and consumer crashes, re-processing will take time

• If visibility timeout is too low (seconds), we may get duplicates

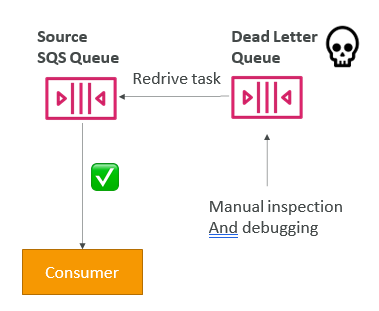
**Amazon SQS – Dead Letter Queue**

* If a consumer fails to process a message within the Visibility Timeout…
* the message goes back to the queue!
* We can set a threshold of how many times a message can go back to the queue
* After the MaximumReceives threshold is exceeded, the message goes into a dead letter queue (DLQ)
* Useful for debugging!
* Make sure to process the messages in the DLQ before they expire:
  + Good to set a retention of 14 days in the DLQ



**SQS DLQ – Redrive to Source**

* Feature to help consume messages in the DLQ to understand what is wrong with them
* When our code is fixed, we can redrive the messages from the DLQ back into the source queue (or any other queue) in batches without writing custom code



**Amazon SQS – Delay Queue**

* Delay a message (consumers don’t see it immediately) up to 15 minutes
* Default is 0 seconds (message is available right away)
* Can set a default at queue level
* Can override the default on send using the DelaySeconds parameter



**Amazon SQS – FIFO Queue**

* FIFO = First In First Out (ordering of messages in the queue)



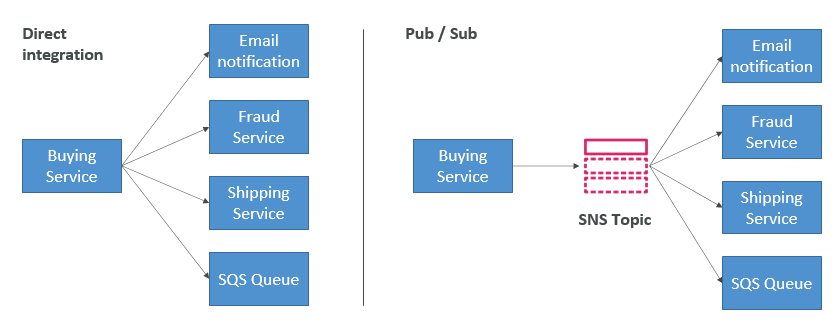
• Limited throughput: 300 msg/s without batching, 3000 msg/s with

• Exactly-once send capability (by removing duplicates)

• Messages are processed in order by the consumer

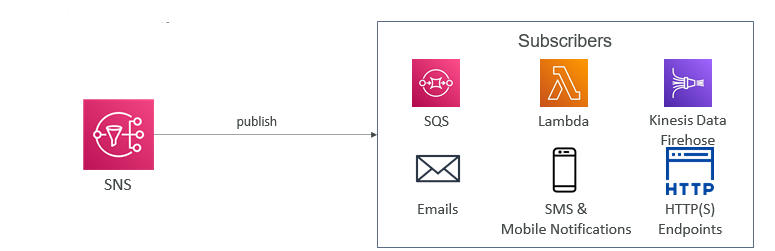
**Amazon SNS**

* What if you want to send one message to many receivers?



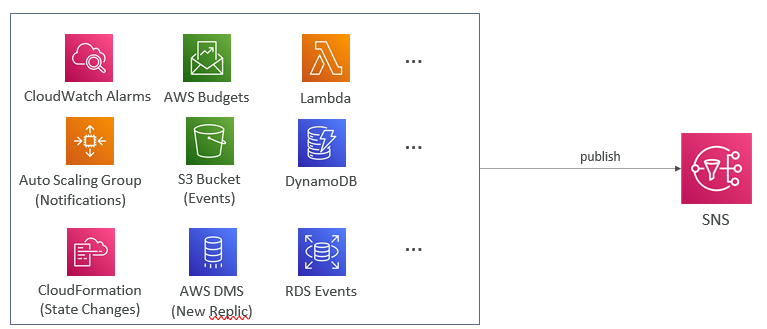
**Amazon SNS**

* The “event producer” only sends message to one SNS topic
* As many “event receivers” (subscriptions) as we want to listen to the SNS topic notifications
* Each subscriber to the topic will get all the messages (note: new feature to filter messages)
* Up to 12,500,000 subscriptions per topic
* 100,000 topics limit



**SNS integrates with a lot of AWS services**

• Many AWS services can send data directly to SNS for notifications



**Amazon SNS – How to publish**

* Topic Publish (using the SDK)
  + Create a topic
  + Create a subscription (or many)
  + Publish to the topic