Decoupling Applications: SQS, SNS, Kinesis, Active MQ

Uday Manchanda

September 17, 2021

1 Intro to Messaging

- Two patterns of application communication
  - Synchronous communications: application to application. Can be problematic if there are sudden spikes in traffic. Best to decouple your applications
  - Asynchronous/event based communications: application to queue to application

2 SQS

2.1 Introduction

- Simple queuing service
- producer sends message to SQS queue
- consumers will poll the messages from the queue, process message, and delete from queue

2.2 Standard Queue

- fully managed service, used to decouple applications
- unlimited throughput, unlimited msgs in queue
- default retention: 4 days, max 14 days
- low latency, at most 256 kb/msg sent
- can have duplicate messages (at least once delivery, occasionally)
- can have out of out of order messages (best effort ordering)

- Producing Messages
  - using the SDK (SendMessage API)
  - message is persisted in SQS until a consumer deletes it
  - EX: send order to be processed
- Consuming Messages
  - running on EC2 instances, servers, lambda, etc
  - polls SQS for messages (receives up to 10 at a time)
  - process the messages (EX: insert msg into RDS db)
  - Delete msg
- Multiple EC2 instances consumers
  - consumers receive and process messages in parallel
  - can scale consumers horizontally to improve throughput of processing
- SQS w/ ASG
  - consumers are running on EC2 instances inside ASG
  - SQS polls for messages
  - ASG needs to scale on some kind of metric - Queue length (CW metric), ApproximateNumberOfMessages
  - Set up CW alarm to check if that metric is breached, will then tell ASG to scale up
- SQS to decouple between application tiers
  - EX: app that processes videos
  - request to process a file is done by sending message to SQS queue
  - Actual video processing done by separate application which receives messages from SQS queue
  - videos are then inserted into S3 bucket
- Security
  - in flight encryption using HTTPS, at rest via KMS keys
  - access controls: IAM policies
  - SQS access policies (similar to S3 bucket policies): useful for cross acct access to SQS queues, for allowing other services (SNS,S3,etc) to write to an SQS queue

2.3 SQS Queue Access Policy

- Cross Account Access
- Publish S3 Event notifications to SQS queue
  - upload object to s3 bucket
  - send msg to SQS queue

2.4 Message Visibility Timeout

- After a msg is polled by a consumer, it becomes invisible to other consumers
- by default, a message visibility timeout is 30 seconds. Means the message has 30 seconds to be processed

2.5 Dead Letter Queue

- EX: if a consumer fails to process a msg within the visibility timeout the msg goes back to the queue
- we can set a threshold of how many times a msg can go back into the queue
- if the MaximumREceives threshold is breached, the msg goes into a dead letter queue
- Msgs in DLQ will eventually expire

2.6 Request-Response Systems

- Requesters (eg multiple producers) send requests into a request queue
- Decouple requester from the responses, you can scale your amount of requests and response this way
- Responders sit in an ASG
- SQS temporary queue client, leverages virtual queues

2.7 Delay Queue

- delay a message up to 15 minutes (default is 0)
- useful if you want to send a per-message delay for every single message

2.8 FIFO Queue

- First in first out, for ordering of messages in the queue
- consumers will poll messages in the same order in which the queue received them
- limited throughput

2.9 SQS + ASG

- ASG with multiple EC2 instances polling for messages from SQS
- eg: more load in queue so more capacity to ec2 instances
- uses CW custom metric, if that metric is breached, it triggers a CW alarm, which tells the ASG to scale
- useful for decoupling application tiers
- EX:
  - EC2 instances behind ASG+ALB receive lots of requests
  - put requests into SQS which is infinitely scalable
  - Have another ASG behind the queue to poll for messages
  - that ASG can scale if necessary

3 SNS

- Send one message to many receivers
- pub sub: publish subscribe
- EX: service publishes messages to SNS topic, SNS topic subscribes to several subscribers
- Only one event producer, but many event subscribers
- Each subscriber will get all messages published
- create topic, create subscription, publish to topic

3.1 SNS+SQS Fan out pattern

- EX: want to send a message to multiple SQS queues, can be risky
- with fan out pattern you push once in SNS topic and then subscribe as many SQS queues as you want to the SNS topic
- fully decoupled, easy to add more SQS queues
- EX: S3 events to multiple queries: use fan out
- SNS FIFO
  - Similar to SQS FIFO
  - if you want to do fan out + ordering + deduplication for example
- Can do message filtering via JSON

4 Kinesis

- Makes it easy to collect, process, and analyze streaming data in real time
- Kinesis data streams: capture, process, and store data streams
- Kinesis data firehose: load data streams into AWS data stores
- Kinesis data analytics: analyze data streams with SQL or Apache FLink
- Kinesis video streams: capture, process, and store video streams

4.1 Kinesis Data Streams

- Streams are made of shards which are numbered
- More shards = more throughput
- Producers send a record (w/ partition key and data) to a kinesis data stream
- Producer might be kinesis producer library or kinesis agent
- Consumers receive records (same partition key, sequence number, and data) from a data stream.
- Consumer might be kinesis data firehose or kinesis data analytics
- Able to reprocess (replay) data
- Data is immutable (cannot be deleted)
- Data that shares the same partition goes to the same shard (ordering)

4.2 Kinesis Data Firehose

- store data in target destinations
- Most common is to read records from kinesis data streams
- Could transform the record via lambda
- Then fill a big batch of data to write that data into a target db/dest
- most common destinations: s3, redshift (copy through S3), or elasticsearch
- could send all or failed data to an s3 backup bucket
- fully managed, serverless, near real time

4.2.1 Kinesis Data Streams vs Firehose

- Data Streams
  - Streaming service
  - Write custom code to send/consume data
  - real time
  - manage scaling
- Data Firehose
  - Load streaming data
  - fully managd, near real time

4.3 Kinesis Data Analytics

- SQL Application
- Read from sources usually kinesis data streams or kinesis data firehose
- write your own sql statements to process data in real time
- send result to: kinesis data streams or data firehose
- send data to kinesis data stream for your own application
- fully managed, serverless
- can create streams out of the queries
- use cases: time series analytics, real time dashboards, real time metrics

5 Data Ordering for Kinesis vs SQS FIFO

- Ordering data for Kinesis
  - use partition key
  - Data ordered within each shard
  - Better for lots of data that needs to be sent and want to have data ordering per shard
- For SQS: SQS FIFO
  - If you don't use a group id, msgs are consumed in the order they are sent with only one consumer
  - EX: you want to scale the number of consumers but want messages to be grouped when they are related: use group id
  - better if you want to have a dynamic number of consumers based on group IDs

6 SQS vs SNS vs Kinesis

- SQS
  - consumer pull data
  - data is deleted after being consumed
  - can have as many workers as we want
  - ordering is only guaranteed in a FIFO queue
  - no need to provision throughput
- SNS
  - push data to many subscribers
  - data is not persisted
  - pub/sub
  - no need to provision throughput
- Kinesis
  - Standard: pull data
  - Enhanced-fan out: push data
  - Possibility to replay data
  - ordering at shard level
  - need to provision throughput

7 MQ

- SQS/SNS are AWS specific
- Managed apache activeMQ
- useful if you want to migrate to the cloud without re-architecting