Integration & Messaging
- SQS, SNS & Kinesis

일시	2023.04.16. (일)	작성자	김민경

1. SQS, SNS & Kinesis

1) 개요

-app 여러 개 배포 시 ⇒ 정보 & 데이터 공유해야 함

-app 커뮤니케이션 패턴

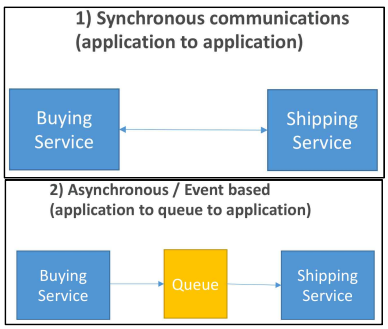
① Synchronous communications (동기 커뮤니케이션)

:app이 다른 app과 직접적으로 연결

② Asynchronous / Event based (비동기 or 이벤트 기반 유형)

:대기열 등으로 불리는 미들웨어가 app들 연결

(직접 소통x기 때문에 비동기적)



-app 간의 동기화 ⇒ 때때로 문제 될 수 0

① 구매가 갑자기 증가 or 한 서비스가 다른 서비스 압도할 경우

② 트래픽 급증 or 예측x 때

∴ app 분리하는 것이 좋음

SQS 사용	대기열(queue) 모델인 경우
SNS 사용	pub/sub 모델인 경우
Kinesis 사용	실시간 스트리밍을 하고 대용량 데이터 다룰 경우

***pub/sub 모델**

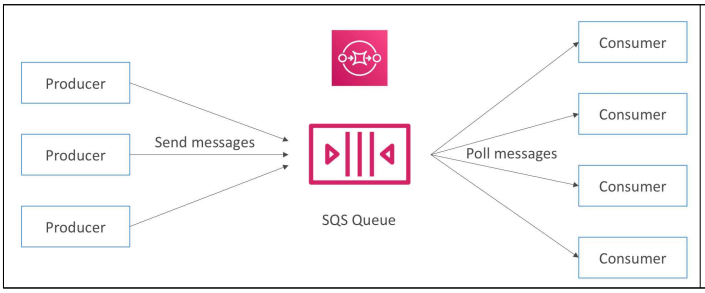
:publish/subscribe 줄임말,
메시지 기반 미들웨어 시스템,
publish(sender), subscriber(receiver)

2. SQS

1) 개요

-간단한 대기 서비스

-대기열⇒ 생산자 & 소비자 사이 분리하는 버퍼 역할



- 생산자⇒ 대기열에 M 보냄
- 대기열⇒ M 처리
- 소비자⇒ 대기열에서 M 폴링
  (대기열에게 소비자 앞으로 온 M 있는지 물어봄)

2) SQS - Standard Queue (표준 대기열용)

***대기열 유형**

① Standard Queue
② FIFO Queue

-가장 오래된 서비스

-완전관리형, app 분리 시 사용

-특성

(1) 무제한 처리량, 대기열의 M 개수 제한x

(2) 보존 기간: 4日(default), 최대 14日

(3) 지연h 짧음(<10ms)

(4) M 작아야 함(256KB 미만)

-중복 M 있을 수 0 (M가 2번 전송되는 경우 존재⇒ ∴적어도 1번 전송됨)

-품절 메시지 보낼 수 0(can have out of order messages(best effort ordering)

메시지 정렬

표준 큐는 메시지 순서를 유지하기 위해 최선을 다하지만 메시지 사본이 두 개 이상 순서대로 전달될 수 있습니다. 시스템에서 해당 순서를 유지해야 하는 경우 FIFO (First-In-First-Out) 대기열을 사용하거나 각 메시지에 시퀀스 정보를 추가하여 메시지가 수신될 때 다시 정렬할 수 있도록 하는 것이 좋습니다.

3) SQS 생산자 (Producer)

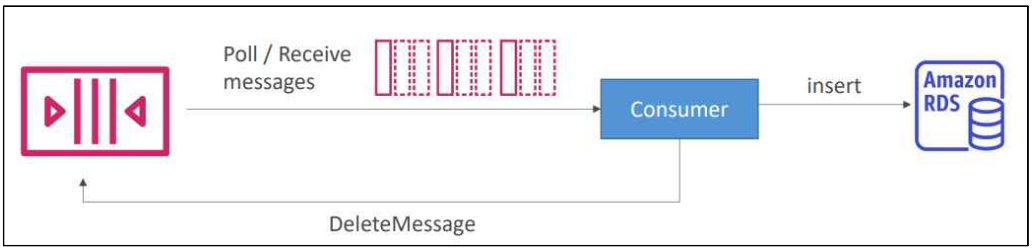
-생산자⇒ SDK 소프트웨어 개발 키트 사용해 SQS에 M 보냄
(SQS에 M 보내는 API⇒ SendMessage API)
-소비자가 M 읽고 삭제할 때까지 SQS 대기열에 유지됨
(M삭제=M처리)
-SQS standard: 무제한 처리량

*SDK
:Software Development Kit,
어떤 sw 만들기 위한 도구 모음,
-개발에 도움이 될 개발 도구 프로그램,
디버깅 프로그램, 문서, API 등이 있음

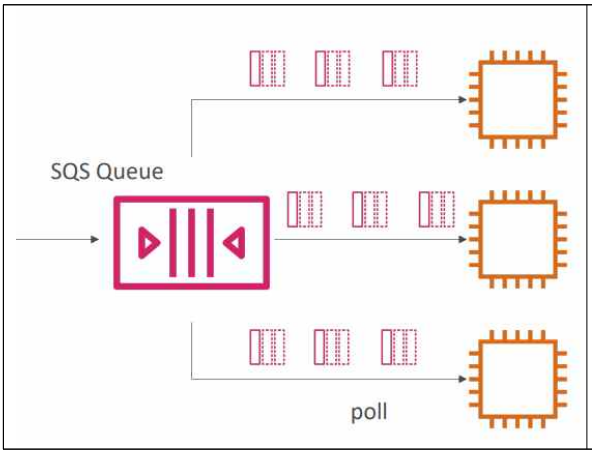
4) SQS 소비자(Consumer)

-app (EC2 인스턴스, 온프레미스 서버, Lambda의 람다 함수에서 실행)
-M 풀링함 (한 번에 최대 10개 M 받을 수0)
-M 처리할 책임
-DeleteMessage API 사용해 대기열에서 M 삭제

*Lambda
:서버 x는 컴퓨터 유형의 서비스

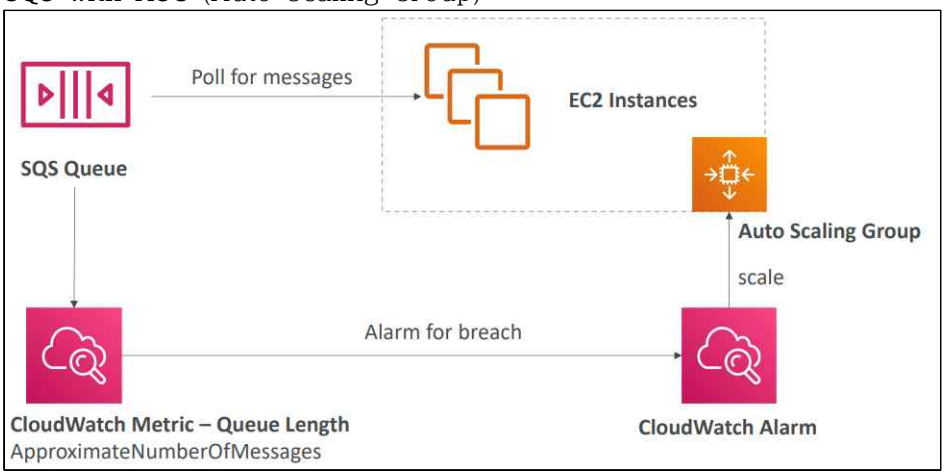


5) Multiple EC2 Instances Consumers



- 여러 소비자 동시에 가질 수 0
- 각 소비자⇒ poll 함수 호출해 다른 M 세트 수신
- M⇒ 적어도 한 번 전송
  (M가 소비자에 의해 빨리 처리 x되면 다른 소비자가 수신)
- 처리량 개선 가능
  (대기열에 多 M가 있을 때 처리량 늘릴 수 0
  →소비자 추가 & 수평 확장)

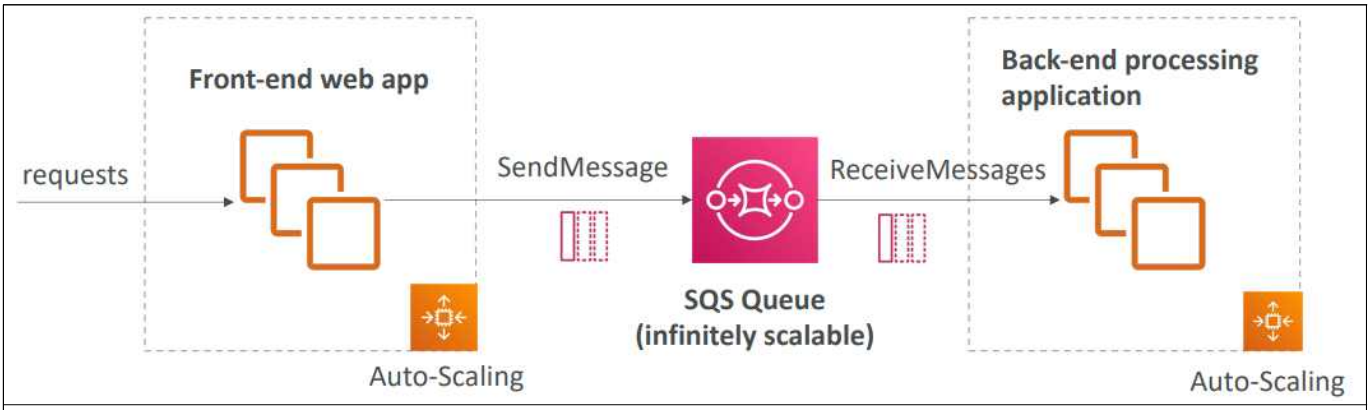
6) SQS with ASG (Auto Scaling Group)



- 소비자⇒ ASG 내부에서 EC2 실행
- ASG⇒ 자동으로 대기열의 길이에 따라 확장
  *CloudWatch의 ApproximateNumberOfMessages 지표: 대기열의 길이, 대기열에 M가 몇개 남아있는지 표시
- 대기열의 길이 특정 수준 넘어갈 시
  ⇒ CloudWatch Alarm 설정해 ASG 용량 ⇑

7) app 계층 분리 (decouple between app tiers)

-SQS⇒ app 계층 분리 시 사용



- 프론트엔드⇒ 파일 처리 요청 받고, 실제 처리까지 하면 시간이 多걸림 & 웹사이트 속도 또한 느려질 수 0

∴ app 계층 분리(파일 처리 요청 & 실체 파일 처리⇒ 다른 app에서 발생하게끔)

- SQS 대기열⇒ 무제한 처리량

∴ 안전, 강력, 확장 가능한 아키텍쳐

- 분리 or 급격히 ⇑한 로드, 시간 초과 등 문제에서 신속한 스케일링 필요한 경우

8) 보안

-암호화

- HTTPS API 사용해 비행 중 암호화
- KMS 키 사용해 미사용 암호 얻음
- 클라이언트 측 암호화 (자체적으로 암호화 및 암호해독)

⇒ SQS에서 기본적으로 지원하는 건 x

*KMS
:Key Manangement Service,
암호화 키 생성, 관리, 제어

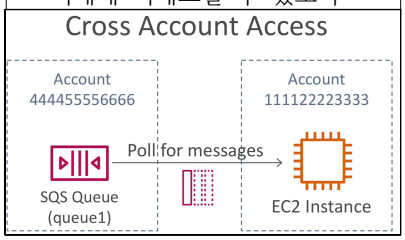
-액세스 제어

- IAM 정책⇒ SQS API 액세스 규제

-SQS 액세스 정책 (÷s3 bucket 정책)

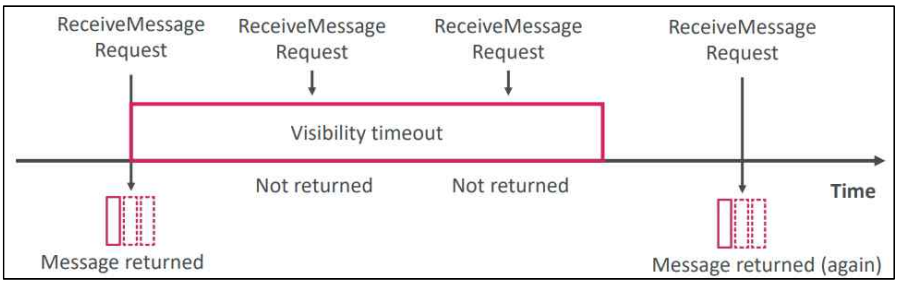
- SQS 대기열에 대한 교차 계정 액세스를 수행하려는 경우
- 다른 서비스(SNS, S3)⇒ SQS 대기열에 쓸 수 있도록 허용하는 경우

*S3 버킷 정책
:적절한 권한을 가진 사용자만
객체에 액세스할 수 있도록



9) Message Visibility Timeout (메시지 가시성 시간 초과)

-대기열에서 수신된 메시지를 볼 수 없는 시간 간격을 지정하는 방법
(가시성 시간 초과 기간 내⇒ 다른 소비자에게 M 보이지 x)



-Default 30s (30s동안 M 처리되어야 함)

-가시성 시간 초과가 경과되고, M 삭제되지 x으면⇒ M는 대기열에 다시 넣음
& 다시 ReceiveMessage API 호출 시⇒ 그 M 받게 됨 (다른 소비지는 보이지x)

-가시성 시간 초과 내에 M 처리x ⇒ M 2번 처리 가능성 (∵2명의 소비자가 수신 or 동일 소비자가 2번 수신)

∴ M 2번 처리하고 싶지 x다면⇒ ChangeMessageVisibility API 호출해 SQS에 알려야 함
(제한 h 단축 or 늘리기 위해⇒ ChangeMessageVisibility 사용하여 새 제한 시간 값 지정)

h으로 설정	대기열에 보이기까지 몇h 걸림
s로 설정	M 처리h 충분 x아 M 중복 처리될 수도 0

10) Long Polling

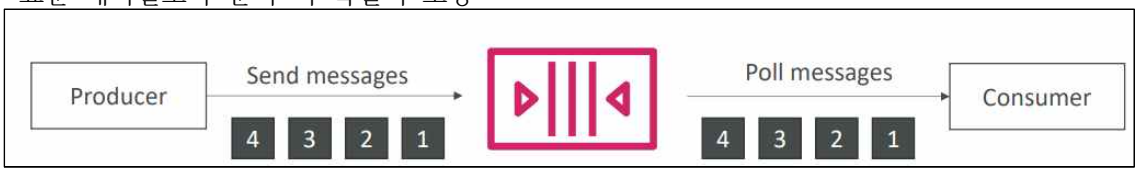
-응답할 M가 x는 경우 바로 응답하지 x고, 대기열에 M가 들어올 때까지 기다렸다가 응답
(소비자 폴링 시 대기열이 비어있으면 기다렸다가,
M가 대기열에 도착하면 롱 폴링 중일 시 자동으로 소비자에게 전달)

*Short Polling
:대기열에 대기하는 M가 x어도 쿼리에 응답

-사용 이유
① 지연h ⇓위해
② 보내는 API 호출 수 ⇓위해
-app의 효율성 & 대기h ⇑
-1~20s 구성 가능
-선호: Long Polling > Short Polling
-구성 방법
① 대기열 레벨에서 구성해 폴링하는 아무 소비자로부터 롱 폴링 활성화
② WaitTimeSeconds 지정해 소비자가 스스로 롤 폴링 하도록 선택

11) SQS - FIFO Queue

-First In First Out (선입선출)
-표준 대기열보다 순서 더 확실히 보장

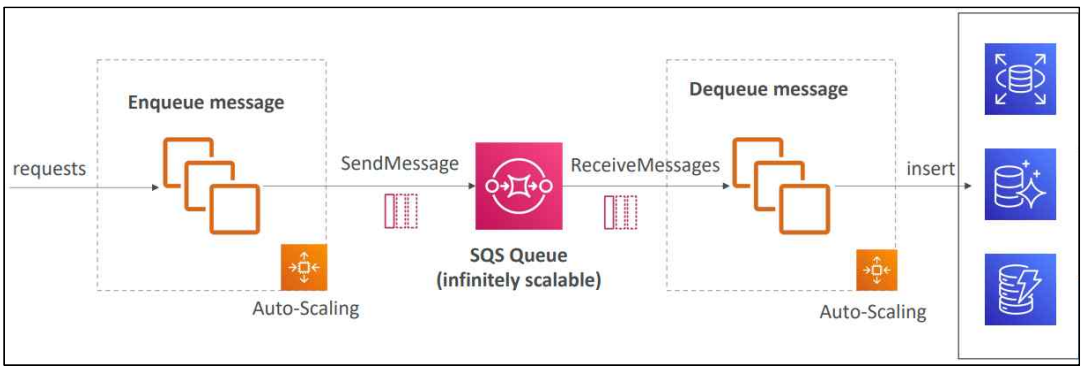


-대기열 처리량: 제한적 (묶음⇒ 초당 3,000개 M, 묶음x⇒ 초당 300개 M처리)
-정확하게 한 번만 전송 0 (중복 제거 기능 통해)
*분리 발생 or M 순서 유지 필요할 때 사용! 처리량 제한 0!

12) SQS - 버퍼로 사용 (SQS as a buffer to DB writes)

*버퍼
:데이터를 한 곳에서 다른 곳으로 전송하는 동안
일시적으로 데이터 보관하는 영역

-로드가 너무 커 트랜잭션에 오류 발생 시
⇒ DB에 바로 요청쓰는 대신 SQS 대기열에 먼저 씀

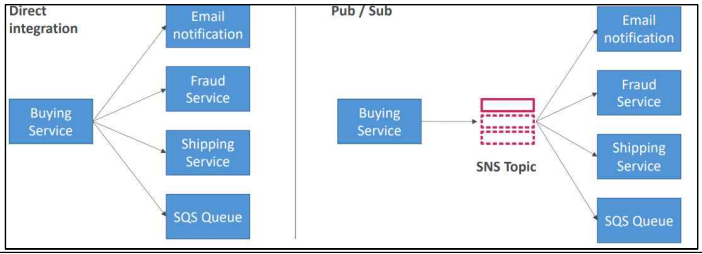


-클라이언트에게 DB에 따로 쓰였다는 확인을 전송할 필요x 때만 사용 가능

3. SNS

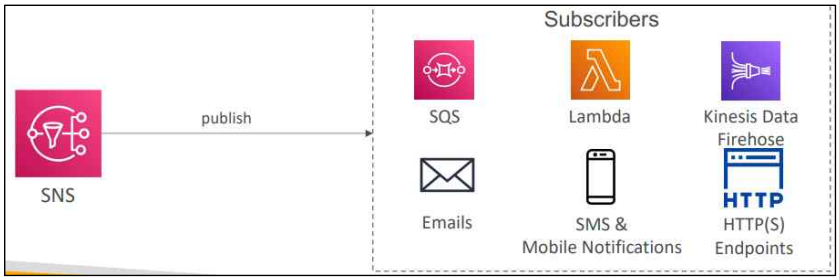
1) 개요

-소비자⇒ SNS 주제를 구독하고 해당 M를 전송받는 것(Pub/Sub)

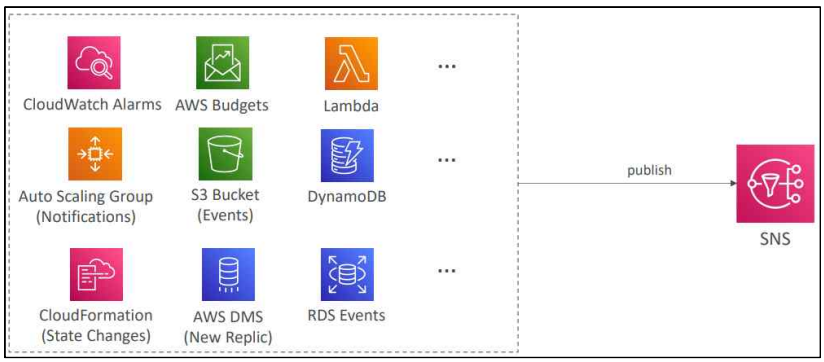


*pub/sub 모델
:publish/subscribe 줄임말,
메시지 기반 미들웨어 시스템,
publish(sender), subscriber(receiver)

-이벤트 생산자⇒ 한 SNS 주제에만 M 보냄
-이벤트 소비자⇒ 해당 주제와 관련한 SNS 알림을 받으려는 사람
-SNS 주제 구독자⇒ 해당 주제로 전송된 M all 받음
-SNS에서 구독자에게 게시할 수 있는 것



-다양한 AWS 서비스와 통합 가능(SNS로 다양한 d보냄)



2) SNS 게시 방법

(1) SDK 주제 게시를 사용해 SNS에 M 게시	(2) 모바일 앱 SDK 전용 직접 게시
① 주제 생성 ② 하나 or 여러 개의 구독 생성 ③ SNS 주제에 게시	① 플랫폼 app 생성 ② 플랫폼 엔드포인트 생성 ③ 플랫폼 엔드포인트에 게시 ④ 수신 가능 대상 : Google GCN, Apple APNS, Amazon ADM 등 ⇒ all 모바일 app으로 알림 수신하게 됨

3) 보안 (= SQS)

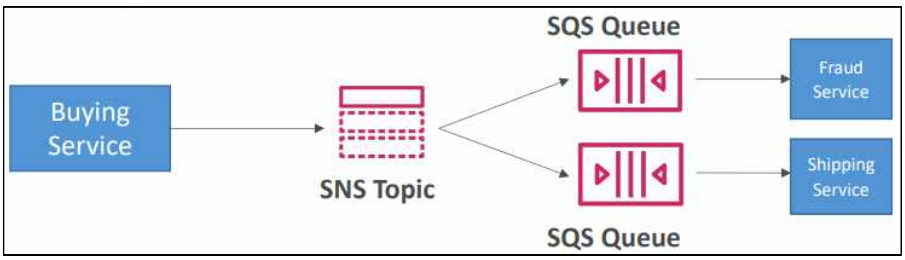
-<u>암호화</u>
- HTTPS API 사용해 비행 중 암호화
- KMS 키 사용해 저장 데이터 암호화
- 클라이언트 측 암호화 (자체적으로 암호화 및 암호해독)

-<u>액세스 제어</u>
- IAM 정책⇒ SNS API 액세스 규제

-<u>SNS 액세스 정책</u> (÷s3 bucket 정책)
- SNS 주제에 대한 교차 계정 액세스를 수행하려는 경우
- 다른 서비스(S3 등)⇒ SNS 주제에 작성할 수 있도록 허용하는 경우

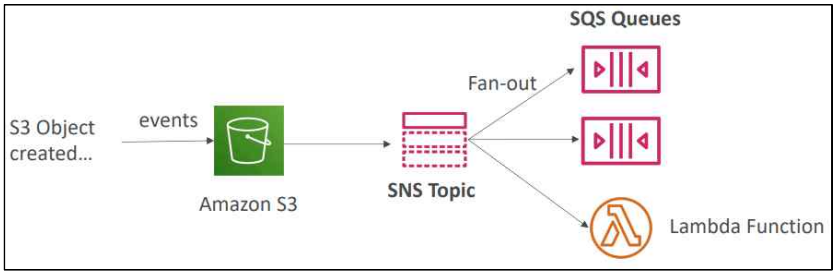
4) SNS + SQS: Fan Out 패턴

-SNS 주제로 M 전송 후 원하는 수의 SQS 대기열들이 이 SNS 주제를 구독하도록 하여
이 대기열들이 SNS로 전송된 모든 M를 수신할 수 있도록
(M를 여러 SQS 대기열로 보낼 때 개별적으로 보내면 문제 발생 가능⇒ Fan Out 패턴 사용)
-완전히 분리된 모델, 데이터 손실 x
-데이터 지속성, 지연처리 수행 0



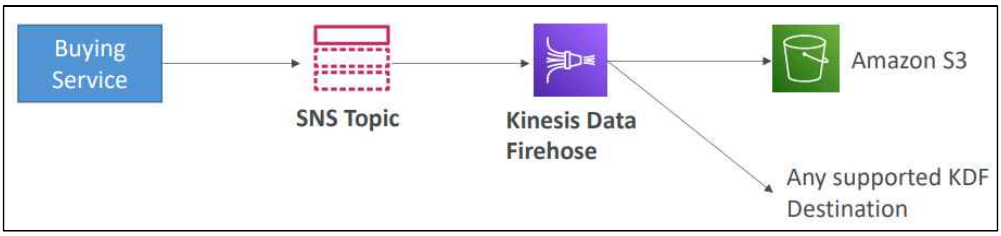
-SNS 주제 구독하도록 더 多 SQS 대기열 추가 가능
-SQS 액세스 정책⇒ SNS 주제가 SQS 대기열에 쓰기 작업하도록 허용해야 함
-리전 간 전달 가능
: 보안 상 가능하면 한 리전의 SNS 주제→ 다른 리전의 SQS 대기열로 M 보낼 수 0

(1) APP: S3 Events to multiple queues
-한 S3 이벤트 규칙에 대한 제한 조건 존재
(객체 생성 같은 이벤트 유형 & /images 와 같은 접두사 조합 동일)
-여러 대기열에 동일한 S3 이벤트 알림 보내고 싶을 때⇒ 팬 아웃 패턴 사용



-S3 객체 생성 & S3 버킷에 이벤트 형성
→이벤트를 SNS 주제로 전송
→팬 아웃 패턴으로 多 SQS 대기열이 SNS 주제를 구독하게 함 (구독자: app, email, Lambda 함수 등)
(팬 아웃 패턴 덕분에 S3에서 발생하는 이벤트의 M가 여러 다른 목적지에 도달 0)

(2) APP: SNS to S3 through KDF (Kinesis Data Firehose)
-KDF 통해 SNS에서 S3로 직접 데이터 전송 가능

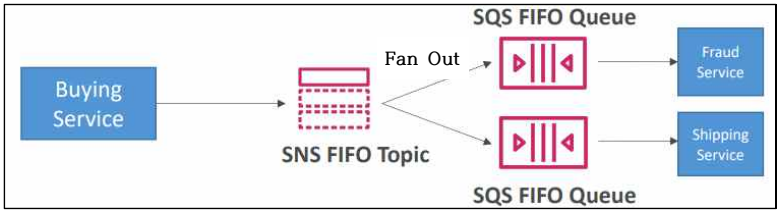


(3) SNS FIFO + SQS FIFO

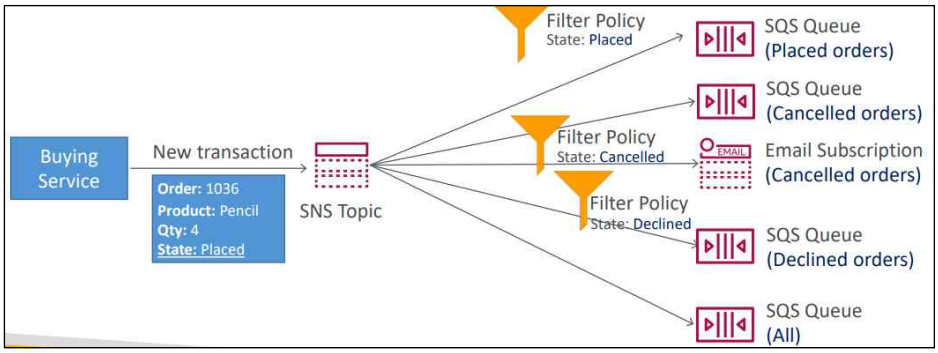
*** SNS - FIFO Topic**
-선입선출(SNS 주제의 M 순서 지정)

-특징 (÷SQS FIFO)
① M 그룹 ID 따라 순서 매김
② 중복 제거 ID 활용 or 내용 비교해 중복 데이터 제거하며 SQS FIFO 대기열을 구독자로 설정
(SQS FIFO 대기열만 구독 가능)
-처리량 제한적

-SQS FIFO 활용해 팬아웃 수행 시⇒ 팬아웃 + 순서 + 중복 제거 필요



(4) SNS - Message Filtering



-팬아웃 패턴 + 메시지 필터링 + FIFO 대기열 + FIFO 주제
-SNS 주제 구독 시 전송되는 M 필터링⇒ JSON 정책 사용
-구독에 어떤 필터링 x다면 all M 받음

4. Kinesis

1) 개요

-실시간 스트리밍 데이터를 쉽게 수집, 처리해 분석

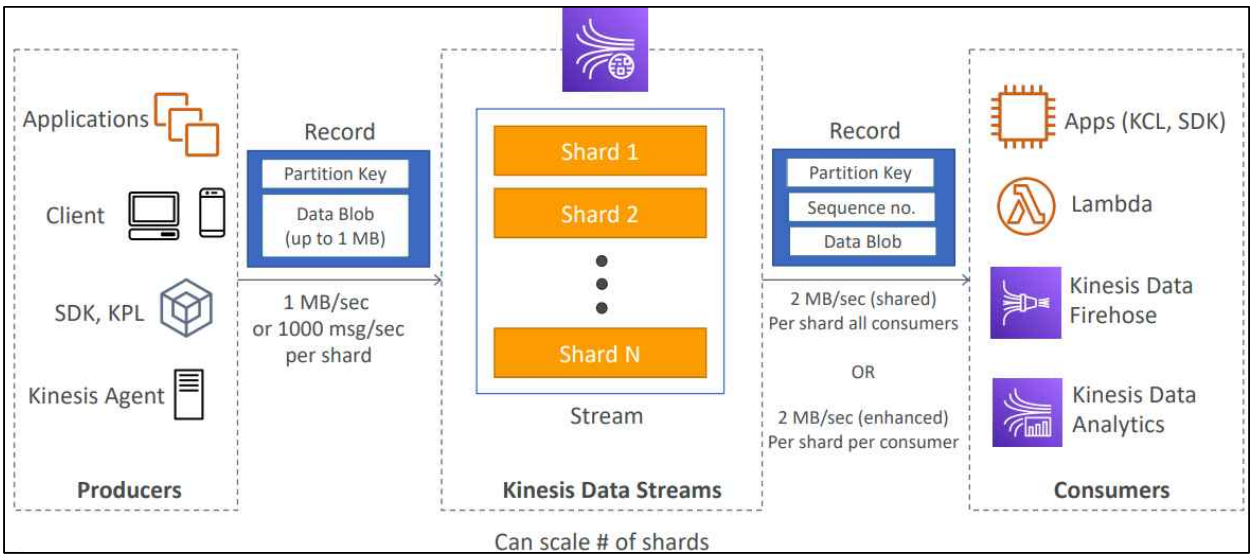
실시간 데이터 : App log, Merics, Website clickstreams, IoT telemetry data 등

-종류

① Kinesis Data Streams: 데이터 스트림 수집, 처리, 저장
② Kinesis Data Firehose: 데이터 스트림을 AWS 내부 or 외부 데이터 저장소로 읽어들임
③ Kinesis Data Analytics: SQL 언어나 Apache Flink를 활용해 데이터 스트림 분석
④ Kinesis Video Streams: 비디오 스트림 수집하고 처리해 저장

2) 종류

(1) **Kinesis Data Streams** : 데이터 스트림 수집, 처리, 저장



(a) 개요

-시스템에서 큰 규모의 데이터 흐름을 다루는 서비스

-샤드⇒ 사전에 프로비저닝 필요 (n개)

⇒ 데이터는 all 샤드에 분배됨

⇒ 스트림의 용량 결정

-생산자⇒ Kinesis Data Streams에 레코드 전달, 데이터는 잠시 KDS에 머물며 여러 소비자에게 읽힘

⇒ 데이터를 스트림으로 보낼 때 초당 1MB 전송 or 샤드당 1s에 1,000개의 M 전송 0

(ex_ 6개 샤드⇒ s당 6MB 얻거나 총 6,000개 M 얻을 수 0)

⇒ AWS SDK, KPL(Kinesis Producer Library), Kinesis Agent 사용해 데이터 전송)

-레코드 구성 요소

파티션 키	레코드가 이용할 샤드를 결정할 때 사용
데이터 블롭	최대 1MB 크기, 값 자체를 의미

-소비자⇒ 다양함

⇒ 레코드 받을 때 파티션 키, 시퀀스 번호(샤드에서 레코드 위치 나타냄), 데이터 블롭

⇒ KCL(Kinesis Client Library), AWS SDK써서 직접 데이터 작성 가능

⇒ AWS Lambda, Kinesis Data Firehose, Kinesis Data Analytics 활용 가능

-소비 유형(KDS에는 여러 소비 유형 존재)

⇒샤드마다 초당 2MB의 처리량을 all 소비자가 공유 or 소비자마다 샤드당 1초에 2MB씩 받을 수 0

(효율성을 높은 팬아웃 방식이라면)

-보존 기간 설정: 1~365日(데이터 재처리 or 확인 가능)

-불변성 (데이터가 Kinesis로 들어오면 삭제x)

-파티션 키가 같은 M들은 같은 샤드로 들어감 (키 기반으로 데이터 정렬)

(b) 용량 모델

① *프로비저닝 모드* (전통적)

-프로비저닝할 샤드 수 정함, API 활용 or 수동 조정

-각 샤드⇒ 초당 1MB or 1,000개의 레코드 받음,

출력량은 초당 2MB(일반적 소비유형 or 효율성⇑ 팬아웃에 적용 가능)

-샤드 프로비저닝 할 때마다 비용 부과

② *온디맨드 모드*

-프로비저닝 필요x (언제든 용량 조정 0)

-초당 4MB or 초당 4,000개 레코드 처리 (default)

-용량⇒ 지난 30日동안 관측한 최대 처리량 기반해 자동 조정

-h당 & 송수신 데이터양 당 비용 부과

*__프로비저닝__: 사용량 사전 예측o
__온디맨드__: 사용량 사전 예측x

(c) 보안

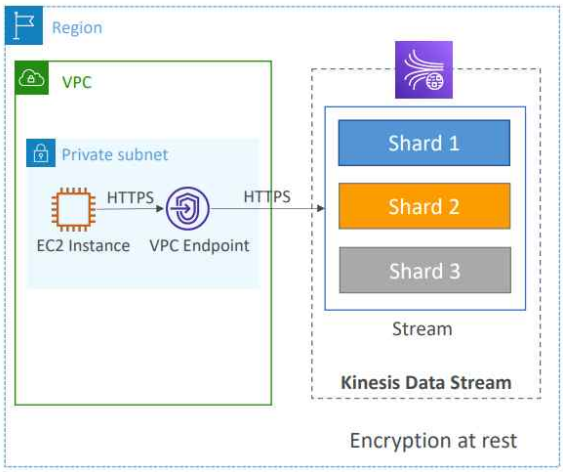
-IAM 정책 사용해 샤드 생성 or 샤드에서 읽어들이는 접근 권한 제어 0

-HTTPS로 전송 중 데이터⇒ 암호화 0,
미사용 데이터⇒ KMS로 암호화

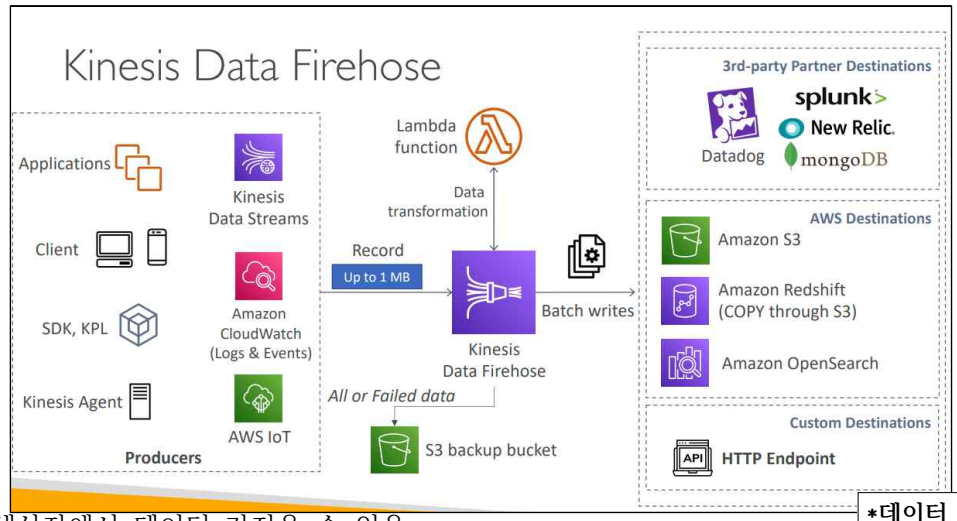
-VPC 엔드포인트 사용 가능

⇒ Kinesis에서 인터넷 거치지 x고
프라이빗 서브넷의 인스턴스에서 직접 손쉽게 접근 0

-all API 요청⇒ CloudTrail로 감시



(2) Kinesis Data Firehose: 데이터 스트림을 AWS 내부 or 외부 데이터 저장소로 읽어들임



-생산자에서 데이터 가져올 수 있음
-생산자가 데이터 전송하면 KDF는 람다 기능 활용해 데이터 변환 가능
-수신처

*데이터 웨어하우스
:의사결정에 도움을 주기 위해 분석 가능한 형태로 변환한 데이터들이 저장되어 있는 중앙 저장소

① AWS 수신처	Amazon S3	-모든 데이터를 S3에 쓸 수 0
	Amazon Redshift	-데이터 웨어하우스 -먼저 S3에 데이터 쓰면 KDF가 복사 명령어 보냄 → 이 복사 명령어가 S3의 데이터를 Redshift로 복사
	Amazon ElasticSearch	-분석 서비스
② 3rd-party Partner 수신처		Datadog splunk> New Relic. mongoDB
③ Custom 수신처		API HTTP Endpoint

-완전 관리형 서비스, 서버리스, 자동 용량 크기 조정
-KDF 통하는 데이터만 비용 지불 (훌륭한 데이터 과금 모델)
-근 실시간 (Near Real Time)
(∵KDF에서 수신처로 데이터를 배치로 써서)
⇒ 전체 배치 x닌 경우: 최소 60s 지연h 발생
or 데이터를 수신처에 보내는데 한 번에 적어도 1MB 의 데이터 있을 때까지 기다려야 함

*배치
: 묶어서 보내는 것

-여러 데이터 형식, 데이터 전환, 변환, 압축 지원
-필요 시 람다 활용해 자체적으로 데이터 변환 가능
-실패한 or all 데이터를 백업 S3 버킷에 보낼 수 0

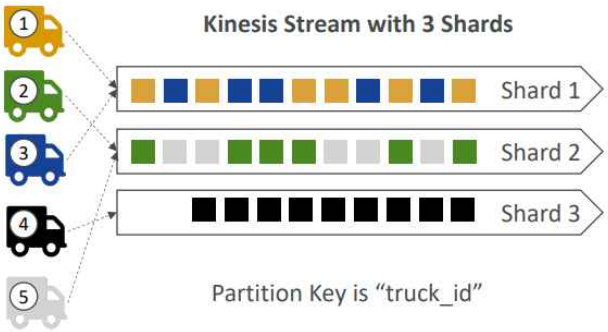
* KDS VS KDF

Kinesis Data Streams	Kinesis Data Firehose
• 데이터 대규모 수집 시 사용하는 스트리밍 서비스 • 생산자 & 소비자에 대해 커스텀 코드 사용0 • 실시간(지연시간 발생, ~200ms) • 용량 조정 가능 • 비용: 제공한 용량만큼 지불 • 데이터 저장: 1~365日 • 여러 소비자가 같은 스트림에서 읽어올 수 있고, 반복 기능 지원(여러 소비자가 반복해서 읽을 수 0)	• 수집 서비스, 데이터를 아래의 것들로 스트리밍함 (Load streaming data into S3 / Redshift / OpenSearch / 3rd party / custom HTTP) • 완전 관리형, 서버리스 • 근 실시간(60sec) • 자동 용량 조정 • 비용: KDF로 통하는 데이터에 대해서만 지불 • 데이터 스토리지 x ∴ 데이터 반복 기능 지원 x

* Kinesis & SQS에서 데이터가 어떻게 정렬되는지

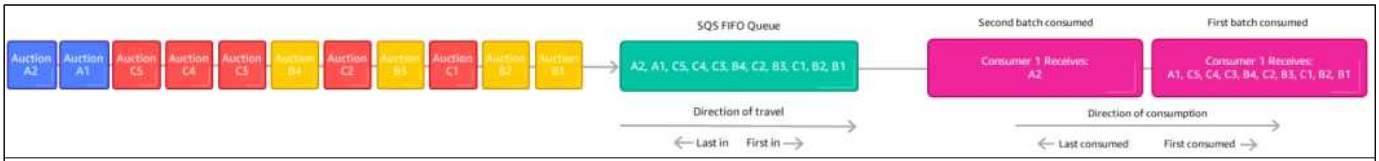
1) Kinesis

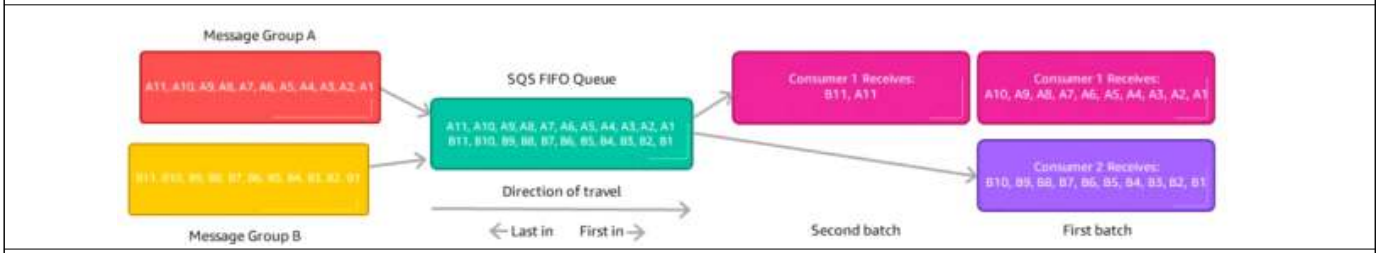
***해시**
:특정한 데이터를 특정한 짧은 길이의 데이터로 변환하는 행위



- 같은 파티션 키 지정 시→ 언제나 동일한 샤드로 전달
  ex_ 트럭 1,3⇒ 언제나 샤드 1에 데이터 전송
- 트럭 & 샤드⇒ 직접 연결x,
  Kinesis가 파티션 키 해시해 어느 샤드로 보낼지 결정

2) SQS





- SQS FIFO 사용
- 그룹 ID 사용x ⇒ 소비자는 오직 하나 (2 배치의 M 소비)
  그룹 ID 사용0 ⇒ 소비자 多
  (각 소비자마다 독자적으로 읽기 가능)
- 그룹 ID 多을수록⇒ 소비자 多짐

***KDS vs SQS FIFO**

-100개 트럭, 5개 Kinesis 샤드, 1개 SQS FIFO라 가정

KDS	SQS FIFO
• 평균적으로 샤드당 트럭 20개 • 트럭 데이터⇒ 각 하나의 샤드에 순서대로 정렬 • 동시에 가질 수 있는 최대 소비자 5개 (∵샤드 5개, 샤드마다 하나의 소비자 필요해서) • 처리량 꽤 多	• 대기열 only 하나 • 소비자 최대 100개 될 수 0 (∵트럭 100개, 각 트럭 ID에 상응하는 그룹 ID 100개 생성) • 최대 초당 300개 M (배치 사용 시 3,000개 M)

5. SQS vs SNS vs Kinesis

SQS	SNS	Kinesis
• 소비자가 데이터 pull • 데이터 처리후 삭제 필요 (다른 소비자가 읽지 x하게) • 소비자 제한 x • 처리량 프로비저닝 필요 x (∵관리형 서비스) • 순서보장⇒ FIFO 대기열 활성화 • 각 M 지연 기능 0 (일정 h 후 대기열에 나타남)	• pub/sub 모델 • 소비자에게 데이터 push (M 복사본 받음) • SNS 주제별 1,250만명 구독자 • 데이터 지속x (제대로 전송x되면 데이터 손실0) • 최대 10만개 주제 • SQS 팬아웃과 통합,	• 소비모드 ① Standard: 데이터 pull ② 향상된 팬아웃: 데이터 push • 데이터 지속(KDS) → 데이터 다시 재생 0 • 실시간 빅데이터 분석, ETL 등에 활용 • 원하는 샤드 양 지정

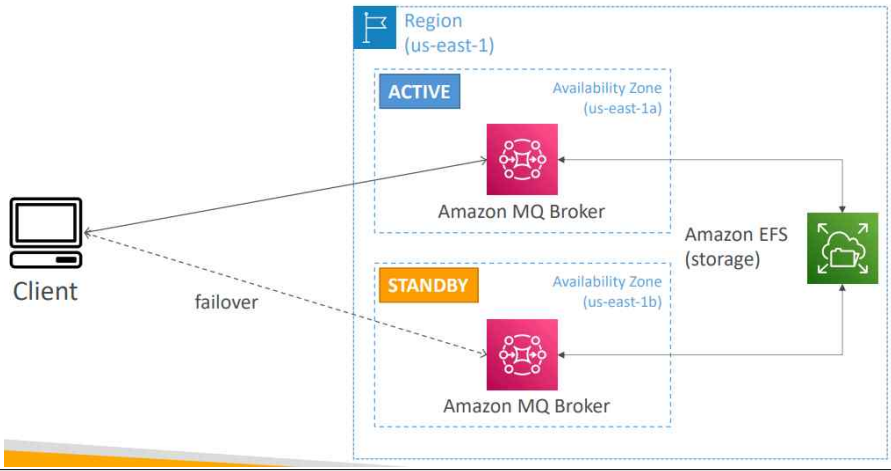
	SNS FIFO 주제⇒SQS FIFO 대기열 결합 0	• 용량 모드 ① 프로비저닝: 미리 샤드양 지정 ② 온디멘드: 샤드양 자동 조정

6. Amazon MQ

1) 개요

-SQS, SNS⇒ 클라우드 네이티브 서비스 (각자 사용하는 API 세트 존재)
-온프레미스에서 기존 app 실행 시⇒ 개방형 프로트콜 사용 (MQTT, AMQP, STOMP, Openwire, WSS 등)
-app→클라우드 마이그레이션
⇒ Amazon MQ 사용 (∵각자 다른 API 사용하여 app 다시 구축→ 싫어서)
-RabbitMQ, ACTIVEMQ를 위한 관리형 메시지 브로커 서비스
(RabbitMQ, ACTIVEMQ⇒ 온프레미스 기술, 개방형 프로토콜 액세스 제공,
Amazon MQ 이용 시 해당 브로커의 관리형 버전을 클라우드에서 사용 가능)
-SQS, SNS처럼 확장성 크지x
-서버에서 실행
but 서버 문제 발생 가능성 0 → HA 위해 장애 조치 & 다중 AZ 설정 실행 0
-SQS처럼 보이는 대기열 기능 & SNS처럼 보이는 주제 기능 제공

2) HA



• 두 영역에 활성, 대기 상태인 Amazon MQ Broker 추가
• 장애 조치 위해⇒ 백엔드 스토리지에 EFS 정의
(EFS⇒ 다중 AZ에 마운트(연결))
• 장애 조치 일어날 때마다⇒ 대기 상태 영역 또한 EFS에 마운트
∴첫번째 활성 대기열과 동일 데이터 가짐, 장애조치도 실행