Serverless

일시	2023.04.23. (일)	**작성자**	김민경

1. Serverless

1) 개요

-서버리스: 서버가 없다 (x)

서버를 관리할 필요가 없다 or 서버가 보이지 않거나 서버를 프로비저닝 하지 x는 것 (o)

-코드/함수를 배치하는 것

-과거: 서버리스는 FaaS(Function as a Service)의 뜻 ⇝ 현재: 더 多것을 의미

-원격 관리되는 것을 모두 포함함(DB, 메시징, 스토리지 등 서버를 프로비저닝 하지 x는 all 것들을 포함함)

-AWS의 서버리스 서비스

• AWS Lambda • DynamoDB • AWS Cognito • AWS API Gateway • Amazon S3	• AWS SNS & SQS • AWS Kinesis Data Firehose • Aurora Serverless • Step Functions • Fargate

2. Lambda

1) 개요

-이벤트에 대한 응답으로 코드를 실행하고 자동으로 기본 컴퓨팅 리소스를 관리하는 서버리스 컴퓨팅 서비스

-EC2 vs Lambda

EC2	Lambda
• 클라우드에서의 가상 서버 • 프로비저닝 필요 ∴프로비저닝 할 메모리 & CPU 크기 제한 • 오토스케일링 그룹으로 스케일링 가능 (자동으로 서버 추가/제거하는 작업해야 함)	• 가상의 함수 -관리할 서버 x이 코드를 프로비저닝하면 함수가 실행됨 • 제한 시간 0 - 짧은 실행 시간 • 온디맨드로 실행 (람다 사용x → 람다 함수 실행x) • 함수가 실행되는 동안만 비용 청구, 호출 받으면 온디맨드로 실행됨 • 스케일링 자동화 (필요시 람다 함수⇑)

2) Benefits (이점)

-가격 책정 쉬움

• 람다 실행된 h만큼 청구

-다양한 AWS 서비스와 통합

-여러 프로그래밍 언어 사용 가능

-CloudWatch와의 모니터링 통합 쉬움

***프로비저닝**

:사용자의 요구에 맞게 시스템 자원을 할당, 배치, 배포해 두었다가 필요시 시스템을 즉시 사용할 수 있는 상태로 미리 준비해 두는 것

-함수 당 더 多 리소스를 프로비저닝 하려면⇒ 함수당 최대 10GB의 RAM 프로비저닝 가능

-함수의 RAM ⇑시키면 → CPU & net의 품질과 성능도 ⇑

3) Lambda를 지원하는 언어

-다양한 언어 지원

- Node.js (JavaScript)
- Python
- Java (Java 8 compatible)
- C# (.NET Core)
- Golang
- C# / Powershell
- Ruby
- Custom Runtime API (community supported, example Rust)

***람다 컨테이너 이미지**

:기계 학습 또는 데이터 집약적 워크로드와 같이 상당한 종속성이 수반되는 대규모 워크로드를 쉽게 구축하고 배포 가능, 운영 편의성, 자동확장, 고가용성, 여러 서비스와 통합 가능하다는 이점

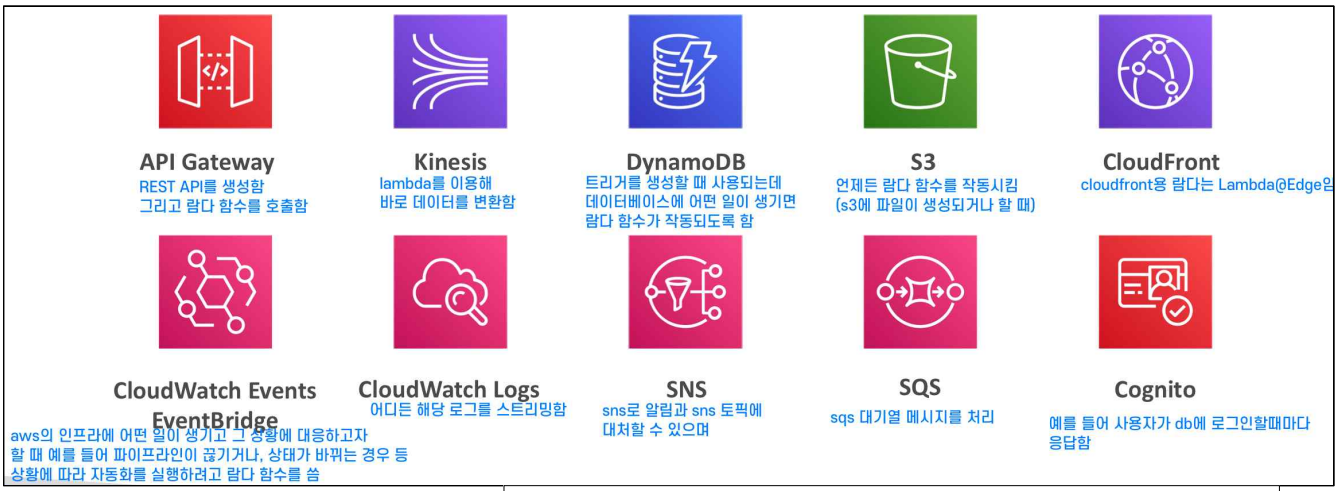
-Lambda Container Image 지원

- Lambda Container Image 자체가 Lambda의 API를 구현해야 함
- ECS / Fargate⇒ 계속 임의의 도커 이미지를 실행할 때 더 多 사용됨

∴ Lambda Container를 실행해야 할 경우

⇒ 해당 Container가 Lambda 런타임 API를 구현하지x으면 ECS or Fargate에서 실행해야 함

4) Lambda와 통합하는 서비스

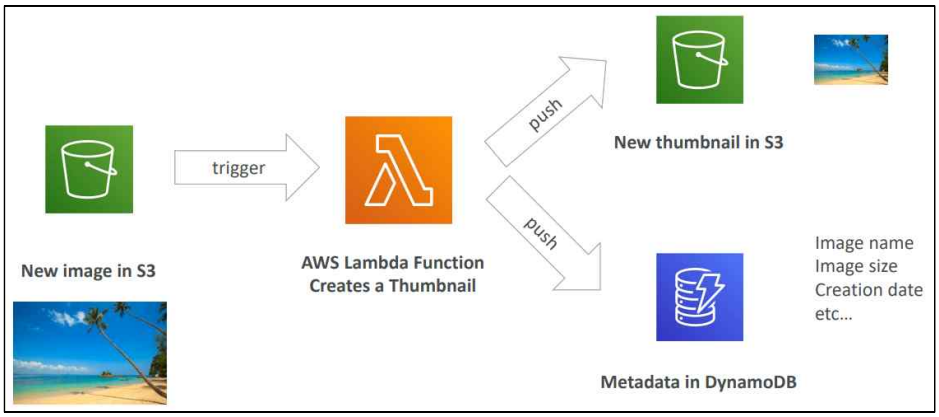


***트리거**

:어떤 사건의 빌미, 또는 시작, 사건이 발생하게 일어나게 하는 계기

5) Lambda 함수 응용

(1) 서버리스 썸네일 생성

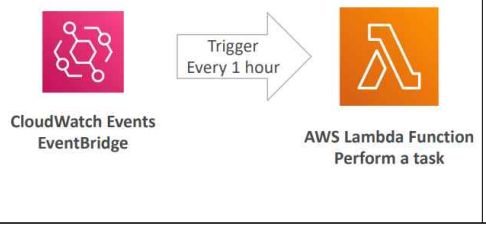


- S3에 새 이미지 업로드→ S3 이벤트 알림을 통해 람다 함수 작동
- 이때 람다 함수에는 썸네일을 생성하는 코드가 있음
- 해당 썸네일⇒ 동일 or 다른 S3 버킷으로 push (원래보다 小 크기로 업로드)
- 람다 함수⇒ 몇 데이터를 DynamoDB에 삽입 가능

(이미지의 메타데이터(ex_이미지 이름, 크기, 생성날짜)가 삽입,
람다 덕분에 기능 자동화)

(2) 서버리스 CRON Job

-CRON: EC2 인스턴스에서 작업을 생성하는 방법



<서버리스 CRON 만드는 방법>

- CRON⇒ EC2 인스턴스 등 가상 서버에서 실행해야 함
- but 인스턴스 실행x or CRON이 아무 일을 하지x으면
  → h이 낭비됨

∴ ex) EventBridge 규칙을 만들고 1시간마다 작동되게 설정
→1시간마다 람다 함수와 통합돼 테스크 수행

- EventBridge & 람다 함수 ⇒ all 서버리스

6) Pricing

https://aws.amazon.com/lambda/pricing/

-호출 당 청구
- 처음 1백만 건 요청⇒ 무료,
  이후 1백만 건 요청마다⇒ 20센트 과금 (so cheap)

-기간 당 청구
- 한 달 간 첫 40만 GB-초 동안의 컴퓨팅 시간⇒ 무료
- 이후에는 60만 GB-초 당⇒ 1$ 과금

-람다로 코드 실행 시⇒ 정말 저렴!

∴app 만들 때 널리 쓰임

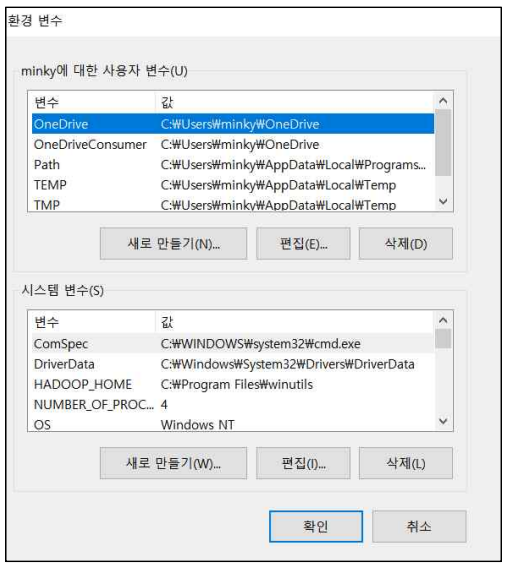
7) 한도

-리전 당 존재

-종류 (2가지)

***메모리 할당**
:프로그램 실행에 필요한 기억 영역을 할당하는 일.
(할당: 몫을 갈라 나누다)

① *Execution* (실행 한도)
- 실행 시 메모리 할당량: 128 MB ~ 10 GB (메모리⇒ 1MB씩 증가)
- 최대 실행 시간: 900s (15m, 초과⇒ 람다 실행에 바람직x)
- 환경변수: 4 KB까지 가질 수 0
- /tmp 폴더: 용량이 있음, 크기는 최대 10GB
- 동시 실행: 1,000개까지 동시 실행 가능
  (증가할 수 0, 미리 예약하는 것이 좋음)



***환경변수**
:시스템의 실행 파일이 놓여 있는 디렉터리의 지정 등 OS 상세서 동작하는 응용소프트웨어가 참조하기 위한 설정이 기록,
즉, 각자 깊숙이 있는 응용프로그램을 쉽게 꺼내쓰기 위해서 미리 변수로 등록해 놓는 것

***tmp 파일**
:temporary 파일의 약자, 임시 파일.
일시적으로 작성, 저장해두는 파일

② *Deployment* (배포 한도)
- 압축 시 최대 크기: 500 MB
- 압축x 시: 250MB
- 시작할 때 크기가 큰 파일 있을 시⇒ /tmp 디렉토리 사용하기
- 배포시 환경변수 한도: 4 KB

8) Customization at the Edge (엣지에서의 사용자 지정)

함수를 사용하여 엣지에서 사용자 지정

PDF | RSS

Amazon CloudFront를 사용하면 자체 코드를 작성하여 CloudFront 배포에서 HTTP 요청 및 응답을 처리하는 방법을 사용자 지정할 수 있습니다. 코드는 최종 사용자(사용자) 가까이에서 실행되어 지연 시간을 최소화하고 서버나 기타 인프라를 관리할 필요가 없습니다. 코드를 작성하여 CloudFront를 통해 흐르는 요청 및 응답을 조작하고, 기본 인증 및 권한 부여를 수행하고, 엣지에서 HTTP 응답을 생성하는 등의 작업을 수행할 수 있습니다.

CloudFront 배포에 작성하고 연결하는 코드를 *엣지 함수*라고 합니다. CloudFront는 엣지 함수를 작성하고 관리하는 두 가지 방법을 제공합니다.

- **CloudFront 함수** – CloudFront 함수를 사용하면 지연 시간에 민감한 대규모 CDN 사용자 지정을 위해 JavaScript로 경량 함수를 작성할 수 있습니다. CloudFront 함수 런타임 환경은 밀리초 미만의 시작 시간을 제공하고 초당 수백만 건의 요청을 처리할 수 있도록 즉시 확장되며 매우 안전합니다. CloudFront 함수는 CloudFront의 기본 기능입니다. 즉, CloudFront 내에서 완전히 코드를 빌드, 테스트 및 배포할 수 있습니다.
- **Lambda@Edge** – Lambda@Edge는 더 가까운 전체 애플리케이션 로직 및 복잡한 함수에 대한 강력하면서도 유연한 서버리스 컴퓨팅을 뷰어에게 제공하는 AWS Lambda의 확장으로, 매우 안전합니다. Lambda@Edge 함수는 Node.js 또는 Python 런타임 환경에서 실행됩니다. 단일 AWS 리전에 함수를 게시하고, 함수를 CloudFront 배포에 연결하면 Lambda@Edge에서 자동으로 전 세계에 코드를 복제합니다.

-보통 함수 & app⇒ 특정 리전에서 배포
but CloudFront 사용 시⇒ 엣지 로케이션 통해 배포,
多 모던 app⇒ app에 도달하기 전에 엣지에서 로직 실행하도록 요구하기도 함

***엣지 로케이션**
:CloudFront를 위한 캐시 서버들의 모음

***로직**
:수많은 논리적 흐름,
데이터를 읽어오거나 저장하고, 이 데이터를 가공하여 사용자에게 의미 있는 정보를 보여주거나 네트워크를 통해 요청을 보내고 받는 행위들은 모두 로직이 모여서 이루어낸 결과물

-Edge Function (엣지 함수): CloudFront 배포에 작성하고 연결하는 코드
  • 목적: 사용자 근처에서 실행해 지연h 최소화
-CloudFront의 2가지 종류의 함수
  ① CloudFront Functions
  ② Lambda@Edge
-Edge Function 사용 시⇒ 서버 관리 필요x (∵전역으로 배포되기 때문)
-Use case: CloudFront의 CDN 콘텐츠를 사용자 지정하는 경우
-사용한 만큼 지불
-완전 서버리스

9) CloudFront Functions & Lambda@Edge Use Cases

-다양한 사용자 지정에 CloudFront Functions & Lambda@Edge가 활용됨

• 웹사이트 보안 & 프라이버시	• 엣지에서의 실시간 이미지 변환
• 엣지에서의 동적 웹 app	• A/B 테스트
• 검진 엔진 최적화(SEO)	• 사용자 인증 및 권한 부여
• 오리진 및 데이터 센터 간 지능형 경로	• 사용자 우선순위 지정
• 엣지에서의 봇 완화	• 사용자 추적 및 분석

10) CloudFront Functions의 활용 & 원리

-CloudFront Functions ⇒ JavaScript로 작성된 경량 함수, 뷰어 요청 & 응답을 수정

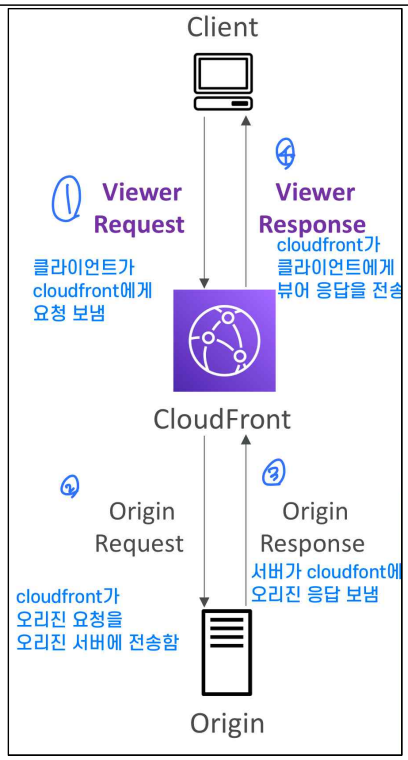
-확장성⇑, 지연 h에 민감한 CDN 사용자 지정에 사용

-시작시간⇒ 1밀리초 미만,
초당 백만 개의 요청 처리

-뷰어 요청 & 응답 수정 시에만 사용

- Viewer Request: CloudFront가 뷰어로부터 요청을 받은 다음 수정 가능
- Viewer Response: CloudFront가 뷰어에게 응답을 보내기 전에 수정 가능

-all 코드가 CloudFront에서 직접 관리됨



11) Lambda@Edge

-CloudFront 함수보다 기능 더 多

-초당 수천개의 요청 처리 가능

-all CloudFront 요청 및 응답 변경 가능

- Viewer Request: =CloudFront
- Origin Request: CloudFront가 오리진에 요청 전송하기 전 수정 가능
- Origin Response: CloudFront가 오리진에서 응답을 받은 후 수정 가능
- Viewer Response: =CloudFront

-함수⇒ us-east-1 리전에만 작성 가능(=CloudFront 배포 관리 리전)

-함수 작성 시⇒ CloudFront가 all 로케이션에 해당 함수 복제

12) CloudFront Fuctions vs Lambda@Edge

	CloudFront Functions	Lambda@Edge
Runtime Support 가장 눈에 띄는 차이점=> 런타임 지원	JavaScript만 지원	Node.js, Python 을 지원
# of Requests	**Millions** of requests per second 확장성은 매우 높음, 수백만 개의 요청을 처리 ⟺	**Thousands** of requests per second 수천개 수준
CloudFront Triggers 트리거 발생 위치	- Viewer Request/Response 뷰어에만 영향력이 있음	- Viewer Request/Response 뷰어와 오리진 모두에게 - Origin Request/Response 영향을 미침
Max. Execution Time 최대 실행 시간	< 1 ms 1밀리초 미만	5 – 10 seconds 실행에 5~10초가 소요됨 이 함수들로 여러 로직을 실행할 수 있음
Max. Memory	2 MB	128 MB up to 10 GB
Total Package Size	10 KB	1 MB – 50 MB
Network Access, File System Access	No	Yes
Access to the Request Body	No	Yes
Pricing	Free tier available, 1/6th price of @Edge	No free tier, charged per request & duration

*Use Cases 차이점

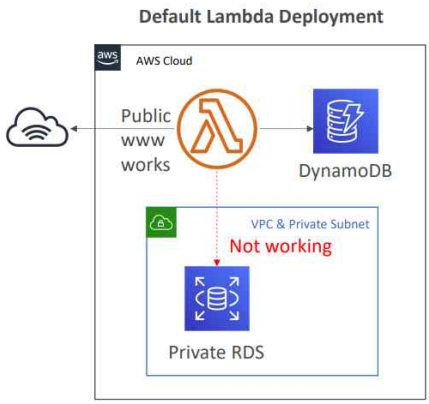
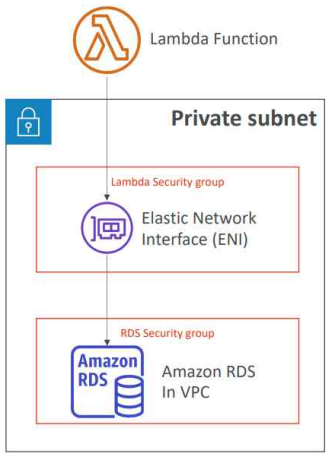
CloudFront Functions	Lambda@Edge
• 캐시 key를 정규화함 ◦ 요청 속성을 변환해 최적의 캐시 키 생성 • 해더 조작 ◦ 요청 or 응답에 HTTP 헤더를 삽입, 수정, 삭제 0 • URL을 다시 쓰거나 리디레션 ◦ 요청 정보에 따라 최종 사용자를 다른 페이지로 리디렉션 or 한 경로에서 다른 경로로 all 요청을	• 실행 시간⇒ longer 때 • CPU or 메모리⇒ 조정 가능 때 • 3rd 라이브러리(타사 라이브러리)에 코드 의존할 때 • 데이터 처리에 외부 서비스를 사용하기 위해 네트워크 액세스가 필요한 함수 • 파일 시스템 액세스 또는 HTTP 요청 본문에 대한 액세스가 필요한 함수

다시 쓸 수 있음 • 요청 권한 부여 ◦ 요청을 허용 or 거부하기 위해 JWT를 생성하거나 검증하는 요청 인증 및 권한 부여에도 사용됨 • all 작업⇒ 1밀리초 이내	

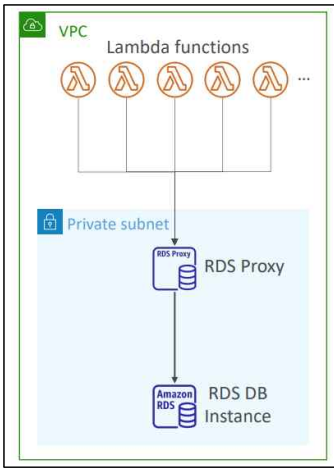
13) Lambda 네트워킹 기초

-람다 함수⇒ 기본적으로 내 VPC 외부에서 시작됨
so VPC 내에서 리소스에 액세스 할 권한x
(RDS, ElastiCache 등의 서비스에 액세스x)

Elastic Network Interface(ENI)는 인스턴스가 AWS 서비스, 다른 인스턴스, 온프레미스 서버, 인터넷 등 다른 네트워크 리소스와 통신할 수 있도록 하

	
<람다 배포의 기본 설정> • 인터넷의 퍼블릭 API에 액세스 가능 (DynamoDB는 퍼블릭 리소스이기 때문에 액세스 가능) • 프라이빗 RDS에 액세스 x ⇒이를 해결하려면 VPC에서 람다 함수 시작하도록	<이를 해결하려면> • VPC ID, 서브넷, sg 지정해야 함 →람다가 서브넷에 ENI 생성해 RDS에 액세스 0

14) Lambda & RDS Proxy

	• VPC에서 람다를 사용하는 대표적인 사례⇒ RDS Proxy • but 람다 함수가 너무 많이 생성되었다가 사라지길 반복할 시 ⇒ 개방된 연결이 너무 多아 RDS의 로드가 ⇑해 시간 초과 문제 발생 ∴RDS Proxy로 연결을 한 곳으로 모아 RDS로의 로드⇓

-장점
• DB 연결의 풀링(수영장, 한번에 모아둠)과 공유를 통해 확장성 향상(람다가 많아져도 RDS는 프록시만 연결해서)
• 장애 발생 시 장애 조치 h ⇒ 66%⇓ (가용성 향상, 연결 보존)
• RDS Proxy 수준에서 IAM 인증을 강제해 보안⇑ 수 있음,
자격증명⇒ Secrets Manager에 저장됨
-람다 함수를 RDS Proxy에 연결하려면⇒ 내 VPC에서 실행해야 함
(RDS Proxy⇒ 퍼블릭 액세스 불가능)

3. Dynamo DB

1) 개요

*트랜젝션
:데이터베이스의 상태를 변화시키기 해서 수행하는 작업의 단위,
질의어(SQL)를 이용하여 DB 접근
(SELECT, INSERT, DELETE, UPDATE)

-완전 관리형 DB, 데이터가 다중 AZ 간 복제됨(가용성⇑)
-NoSQL DB, 관계형 DB x, 트랜젝션 기능 있음
-방대한 워크로드로 확장 가능(∵DB가 내부에서 분산되기 때문)
-초당 수백만 개의 요청 처리, 수조 개의 행, 수백 TB의 스토리지 가짐
-성능⇒ 한 자릿수 밀리초, 일관성⇑
-보안 관련된 all 것(보안, 권한부여, 관리 기능)⇒ IAM과 통합되어 있음
-비용 小, 오토 스케일링 기능 탑재
-유지 관리 or 패치x어도 항상 사용 가능
-테이블로 구성됨, db 생성할 필요x (⇔RDS & Aurora)
-테이블 클래스 종류
① Standard Table Class: 액세스가 빈번한 데이터 저장
② Infrequent Access(IA) Table Class: 액세스가 빈번하지 x은 데이터 저장
-테이블 생성 시⇒ 각 테이블마다 Primary Key 부여됨
-각 테이블에 행(항목) 무한히 추가 가능
-각 항목⇒ 속성 가짐(열에 표시됨, null 가능)
(⇔RDS & Aurora: 열 추가 시 복잡)
-최대 항목 크기: 400 KB (큰 객체 저장에 적합x)
-다양한 데이터 유형 지원

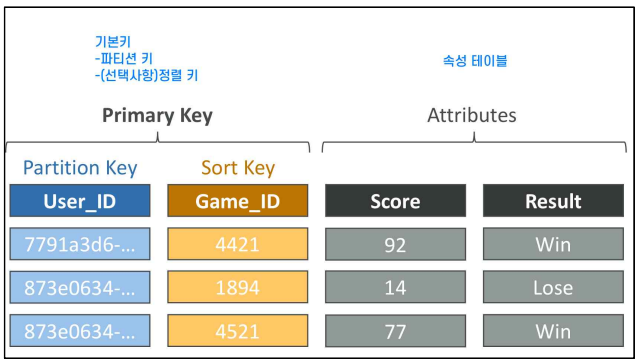
Scalar Types	문자열, 숫자 ,바이너리, 불리언, null
Document Types	목록, 지도
Set Types	문자열/숫자/바이너리 세트

*바이너리
:0과 1, 두 숫자로만 이루어진 이진법

-DynamoDB에서는 스키마를 빠르게 전개할 수 있음

*스키마
:데이터를 DB에 어떤 구조로 저장할 것인가

-테이블 ex)



2) Read/Write Capacity Modes (읽기/쓰기 용량 모드)

-DynamoDB 사용 시 읽기/쓰기 용량 모드 선택해야 함
-테이블 용량 관리 방법을 제어하는 2가지 모드
① *Provisioned Mode* (default)
-로드 예측 가능하고 서서히 전개되며 비용 절감 원할 때 적합
-용량 미리 프로비저닝(초당 읽기/쓰기 요청 수 예측해 미리 테이블 용량 지정)
-프로비저닝 된 RCU(Read Capacity Units:읽기 용량 단위) & WCU(Write Capacity Units:쓰기 용량 단위) 만큼의 비용 지불
-미리 용량 계획해도 오토 스케일링 기능이 있으니 테이블 로드에 따라 자동으로 RCU & WCU 조절 가능
② *On-Demand Mode*
-워크로드 예측할 수 x거나 급격히 ⇑하는 경우 적합
-미리 용량 계획 필요x (RCU & WCU 개념x)
-사용한 만큼 지불, all 읽기 & 쓰기에 비용 지불(more expensive)

3) DAX (DynamoDB Accelerator)

-DynamoDB의 고급 기능

-DynamoDB 응답 시간⇒한 자릿수 밀리초 단위

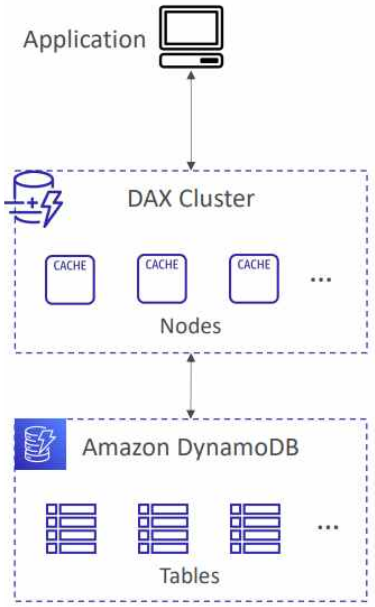
DAX⇒ 캐시 데이터에 마이크로초 단위의 응답 시간이 필요할 때

-고가용성의 완전 관리형 무결절 인메모리 캐시

-기존 DynamoDB와 API 호환됨

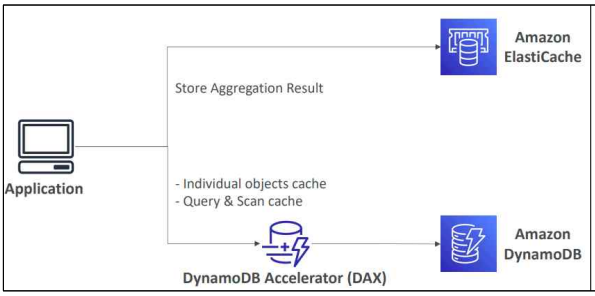
-캐시 TTL(default)⇒ 5m (변경 가능)

***무결절**
:seamless, 끊어짐이 x는 것



- DynamoDB 테이블 & app이 있을 때
  몇몇 캐시 노드가 연결된 DAX 클러스터를 생성하면
  백그라운드에서 DAX 클러스터가 DynamoDB 테이블에 연결됨

***DAX vs ElastiCache**



- DAX⇒ 개별 객체 캐시와 쿼리 & 스캔 캐시 처리 시 유용
- ex) 집계 결과 저장⇒ ElastiCache가 적합
  대용량 연산 저장⇒ DAX가 적합
- 두 서비스⇒ 상호보완적 성격
- DynamoDB에 캐싱 솔루션 추가 시⇒ 보통 DAX 사용

https://dev.classmethod.jp/articles/lim-dynamodb-accelerator/

4) Stream Processing (스트림 처리)

-테이블의 수정 사항(생성, 업데이트, 삭제)을 포함한 스트림 생성 가능

-Use cases

- DynamoDB 테이블 변경사항에 실시간 반응
- 실시간으로 사용 분석
- 파생 테이블 삽입
- 리전 간 복제
- DynamoDB 테이블 변경사항에 대해 람다 함수 실행

***스트림**

○ [데이터 관점]

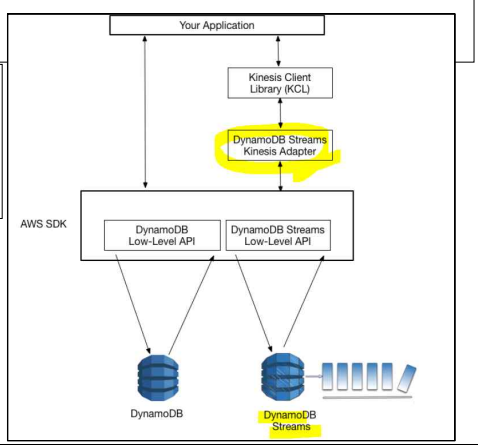
- 일반적으로, 데이터,패킷,비트 등의 일련의 연속성을 갖는 흐름을 의미
  . 음성,영상,텍스트 등의 작은 데이터 조각들이 하나의 줄기를 이루며,
  . 순서대로 물 흐르듯이 전송되는 데이터 열(列)

-스트림 처리 2가지 종류

① *DynamoDB Streams*

- 보존기간 24h
- 소비자 수 제한 됨
- 람다 트리거와 함께 사용하면 좋음,
  자체적 읽기 실행하려면 DynamoDB Stream Kinesis 어댑터 사용

***람다 트리거**
:람다 함수를 실행할 수 있는 이벤트,
ex) S3에 파일이 적재되면 이를 이벤트로 받아
람다 함수를 실행할 수 있음



DynamoDB Streams Kinesis 어댑터를 사용하여 스트림 레코드 처리

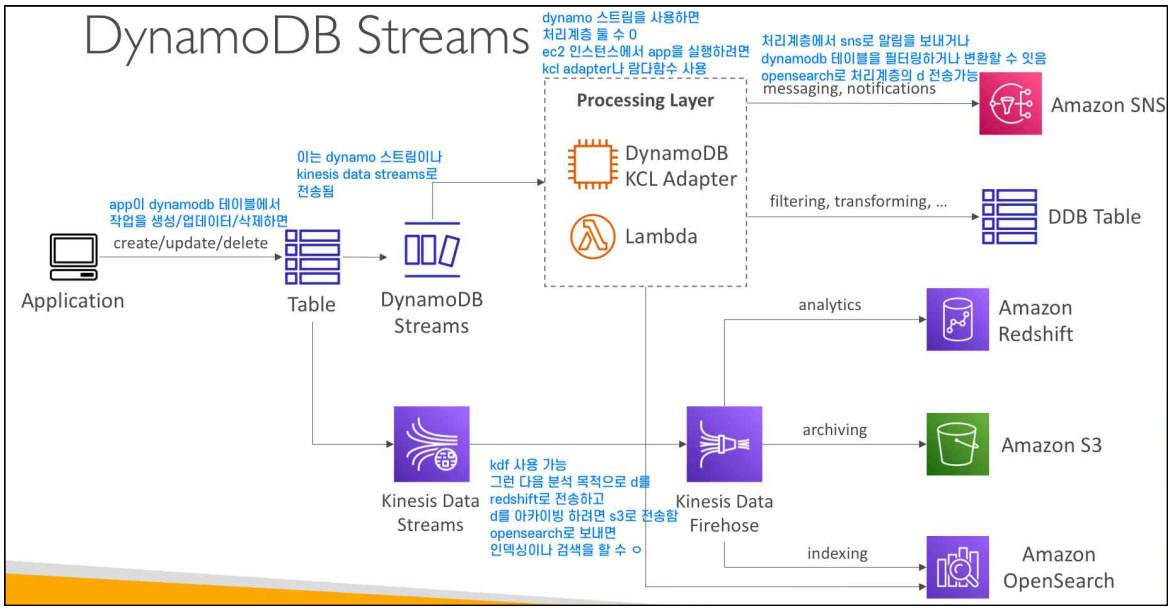
PDF | RSS

Amazon Kinesis 어댑터 사용은 Amazon DynamoDB의 스트림을 소비할 때 권장되는 방법입니다. DynamoDB Streams API는 대규모로 스트리밍 데이터를 실시간으로 처리하는 서비스인 Kinesis Data Streams의 API와 유사합니다. 두 서비스 모두 데이터 스트림이 샤드로 구성되어 있습니다. 샤드란 스트림 레코드의 컨테이너입니다. 두 서비스의 API에는 `ListStreams`, `DescribeStream`, `GetShards` 및 `GetShardIterator` 작업이 포함되어 있습니다. (이러한 DynamoDB Streams 작업은 Kinesis Data Streams의 해당 작업과 유사하지만 100% 동일하지는 않습니다.)

② *Kinesis Data Streams* (newer)

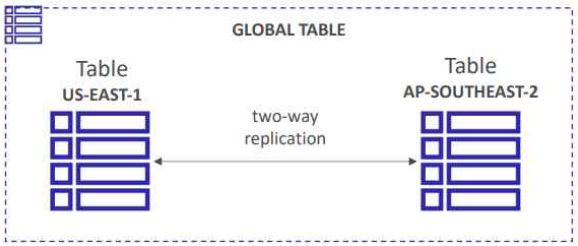
- 보유기간: 1년
- 더 多은 소비자 수 가짐
- 데이터 처리하는 방법이 훨씬 多

Process using AWS Lambda, Kinesis Data Analytics, Kineis Data Firehose, AWS Glue Streaming ETL…



5) Global Tables

-글로벌 테이블: 여러 리전 간 복제 가능한 테이블



- 테이블을 us-east-1과 ap-southeast-2에 둘 수 있음
- 테이블 간 양 방향 복제 가능
- us-east-1 이나 ap-southeast-2 테이블 둘 中 하나에 쓰기를 하면 됨

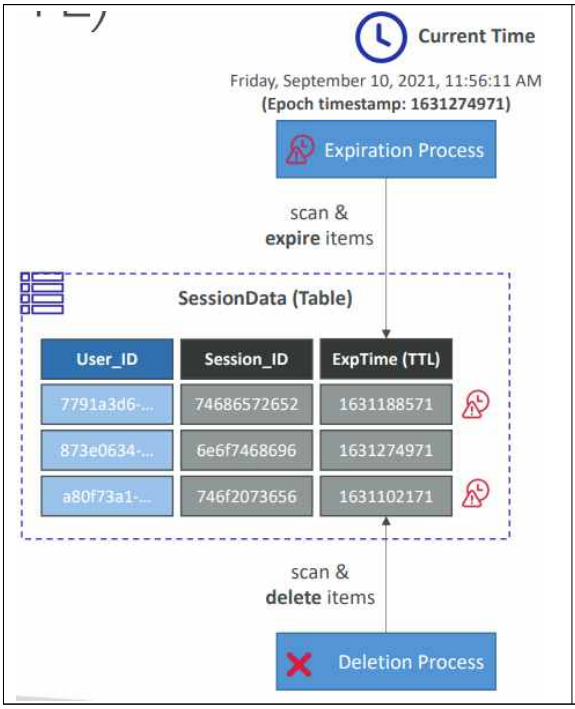
-DynamoDB 글로벌 테이블⇒ 복수의 리전에서 짧은 지연h으로 액세스 가능하게 함

-다중 활성 복제 가능 (app이 all 리전에서 테이블을 READ하고 WRITE 할 수 있음)

-글로벌 테이블 활성화하려면

⇒ DynamoDB 스트림을 활성화해야 리전 간 테이블 복제할 수 있는 기본 인프라 구축됨

6) TTL (Time To Live)



- SessionData 테이블⇒ ExpTime(TTL, 만료기간 속성) 있음, 이 안에 타임 스탬프 들어감
- 타임스탬프 지날 시⇒ 자동으로 항목 삭제
  - TTL 정의 후 Epoch timestamp에서 현재 h이 ExpTime을 넘어간 경우⇒ 해당 항목 삭제 진행

> - TTL 속성은 숫자 데이터 형식을 사용해야 합니다. 다른 데이터 형식(예: 문자열)은 지원되지 않습니다.
> - TTL 속성은 Epoch 시간 형식을 사용해야 합니다. 예를 들어 2019년 10월 28일 13:12:03 UTC의 Epoch 타임스탬프는 1572268323입니다. EpochConverter와 같은 무료 온라인 변환기를 사용하여 올바른 값을 얻을 수 있습니다.
>
> **참고:** 타임스탬프는 밀리초가 아닌 초 단위여야 합니다(예: 1572268323000 대신 1572268323 사용).

-Use cases

- 최근 항목만 저장, 2년 후 데이터 삭제해야 한다는 규정 따라야할 때
- 웹 세션 핸들링(사용자가 웹사이트 로그인 시 해당 세션을 DynamoDB에 2시간 동안 저장 후 자동 삭제)

> ***세션**
> :웹 사이트의 여러 페이지에 걸쳐 사용되는 사용자 정보를 저장하는 방법

7) Backups for disaster recovery (백업 옵션)

-재해복구에도 활용

지정시간 복구(PITR:poinr-in-time recovery) 사용한 지속적 백업	온디멘드 백업
• 활성화 선택 가능, 35일 동안 지속 • 활성화 시 백업 기간 내에 언제든 지정시간 복구 가능 • 복구 진행할 시 새로운 테이블 생성	• 직접 삭제할 때까지 보존됨 • Dynamo의 성능 or 지연h에 영향 주지x • 백업을 더 제대로 관리할 수 있는 AWS Backup 서비스가 있음 ⇒백업에 수명주기 정책 활성화 가능, 재해 복구 목적으로 리전 간 백업 복사 가능 • 복구 진행할 시 새로운 테이블 생성

8) Amazon S3와의 통합

Export to S3	
	• 테이블을 s3로 내보내서 데이터 분석 가능 • PITR 반드시 활성화해야함 → 35일 이내 어떤 시점으로든 테이블 내보낼 수 있음 • 테이블 내보내도 읽기 용량 or 성능에 영향x • 감사 목적으로 스냅샷 확보 가능 • 데이터를 DynamoDB로 다시 가져오기 전 데이터 ETL 등 대규모 변경 실행 가능 • 내보낼 때 DynamoDB JSON or ION 형식 이용

Extract
source database로부터 data를 읽어내는 process

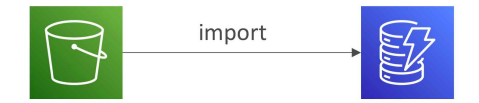
Transform
원래의 source로부터 추출된 data와 최종 원하는 data 구성 및 형태를 연결하는 process로, rule, lookup table, data 결합 및 재구성을 사용한다.

Load
target database에 data를 저장하는 process

ETL Tool
source database(들)로부터 target database(data warehouse, data mart, 다른 database)로 data를 이동, 통합하기 위해 사용되며, 중심 process는 data transformation, cleansing(정제), 중복 제거 및 통합, 재구성이다.

*ETL
:추출(Extract), 변환(Transform), 적재(Load), 한 곳에 저장된 데이터를 필요에 의해 다른 곳으로 이동하는 것

Import to S3



- S3에서 테이블 가져오는 것
- S3에서 CSV, DynamoDB JSON, ION 형식으로 내보낸 다음 새로운 DynamoDB 테이블 생성
  (쓰기 용량 소비하지 x고 테이블 생성)
- 가져올 때 발생한 오류⇒ all CloudWatch Logs에 기록

4. AWS API Gateway

1) 개요

-API를 쉽게 생성, 게시, 유지 관리, 모니터링 및 보호할 수 있는 완전 관리형 서비스
-람다 & API Gateway 통합⇒ 완전 서버리스 app 구축 (∴인프라 관리 필요x)
-WebSocket 프로토콜 지원
-버전 1,2,3dl 생겨도 클라이언트 연결이 끊어지지 x
(∵ API Versioning을 핸들링해서)
-dev, test, prod 등 여러 환경 핸들링
-보안에도 활용 가능
-API 키 생성, API Gateway에 클라이언트 요청 과도하면 요청 스로틀링(제한) 가능
-Swagger / Open API과 같은 공통 표준 사용해 신속히 API 정의해 가져올 수 있음
-요청과 응답 변형하거나 유효성 검사해 올바른 호출 실해오디게 할 수 있음
-SDK & API 스펙 생성 or 응답 캐시 가능

*REST API
:핵심 컨텐츠 및 기능을 외부 사이트에서 활용할 수 있도록 제공되는 인터페이스

2) Integrations (어떤 통합 지원하는지)

① *Lambda function*
-람다 함수 지연 호출
-람다 함수를 사용하는 REST API를 완전 서버리스 app에 노출시키는 간단한 방법

② *HTTP*
-백엔드의 HTTP 엔드포인트 노출시킬 수 있음
-ex) 온프레미스에 HTTP API 있거나 클라우드 환경에 API 활용 시 속도 제한 기능, 캐싱, 자동차 인증, API 키 등의 기능 추가 가능

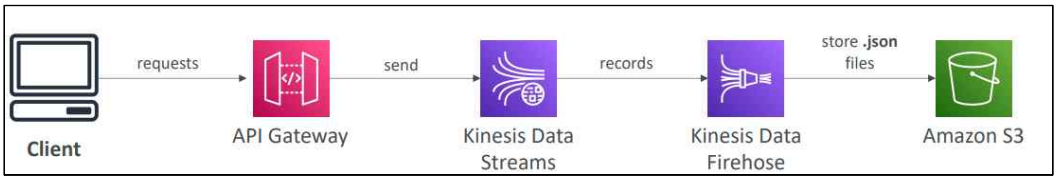
③ *AWS Service*
-어떤 AWS API라도 노출 가능
-ex) 단계 함수 워크플로우 시작 가능, API Gateway에서 직접 SQS에 메시지 게시 가능
-인증 추가 or API를 퍼블릭으로 배포 or 특정 AWS 서비스에 속도 제한 추가하기 위해 통합하는 것임

3) Endpoint Types (배포 방법)

Edge-Optimized (default)	Regional	Private
• 글로벌 클라이언트 용 (전세계 누구나 API 게이트웨이 액세스 가능) • all 요청 ⇒ CloudFront 엣지 로케이션 통해 라우팅됨 (지연h 개선) • API 게이트웨이 ⇒ 여전히 생성된 리전에 위치	• CloudFront 엣지 로케이션 원하지 x때 사용 • all 사용자 ⇒API 게이트웨이 생성한 리전과 같은 리전에 있어야 함 • 자체 CloudFront 배포 생성 가능 ⇒엣지 최적화 배포와 동일한 결과, 캐싱 전략 & CloudFront 설정에 더 多 권한 가질 수 0	• ENI 같은 인터페이스 VPC 앤드포인트 사용해 VPC 내에서만 액세스 가능 • API 게이트웨이에 액세스 정의 시 ⇒리소스 정책 사용

4) 응용- KDS와의 통합

-AWS 서비스에 API 게이트웨이 사용하는 예시 (KDS)



- KDS와 클라이언트 사이에 API Gateway 둠
- 클라이언트가 API Gateway로 HTTP 요청 보내면 KDS에 전송하는 메시지 구성해 전송함 (서버 관리 필요x)

5) Security

(1) 사용자를 식별하는 방법

① IAM 역할 사용

-API Gateway의 API에 액세스 할 때 IAM 역할 사용하는 것

-내부 app에서 유용함

② Cognito

-모바일 app or 웹 app의 외부 사용자에 대한 보안 조치로 사용

Amazon Cognito 소개

Amazon Cognito는 웹 및 모바일 애플리케이션 안에 고객 ID 및 액세스 관리(CIAM) 기능을 구현하는 데 도움이 됩니다. 사용자 인증 및 액세스 제어를 몇 분 안에 빠르게 애플리케이션에 추가할 수 있습니다.

③ Custom Authorizer (사용자 지정 권한 부여자)

-자체 로직 실행하려면 사용자 지정 권한 부여자 사용

***도메인 이름**

:인터넷의 실제 IP 주소와 연결된 기억하기 쉬운 이름

(2) 사용자 지정 도메인 이름과 ACM 통합 (HTTP 보안)

***ACM**

:AWS Cerificate Manager,
AWS 웹 사이트와 애플리케이션을 보호하는 퍼블릭 및 프라이빗 SSL/TLS 인증서와 키를 만들고, 저장하고, 갱신하는 복잡성을 처리

① 엣지 최적화 엔드포인트 사용 시⇒ 인증서는 us-east-1에 있어야 함

② 리전 엔드포인트 사용 시⇒ 인증서는 API Gateway 단계와 동일한 리전에 있어야 함

③ Route 53에 CNAME or A-alias 레코드 설정해 도메인 API Gateway 가리키도록 해야함

5) CNAME vs Alias

-AWS 리소스(Load Balancer, CloudFront...) 사용하는 경우 ⇒ 호스트 이름 노출,
보유한 도메인에 호스트 이름을 매핑하고자 함

① CNAME	-호스트 이름이 다른 호스트 이름 향하도록 -루트 도메인 이름이 x닌 경우에만 가능 -Zone Apex라는 DNS 네임스페이스의 상위 노드로 사용 x
② Alias	-호스트 이름이 특정 AWS 리소스로 향하도록 -루트 & 비루트 도메인 all에 작동 -무료 -자체적 health check -Zone Apex라는 DNS 네임스페이스의 상위 노드로 사용 0 -AWS 리소스를 위한 Alias 레코드 타입은 A or AAAA, 리소스는 IPv4 or IPv6 中 하나 -TTL 설정x (ROUTE 53에 의해 자동 설정) -EC2 DNS name을 타겠할 수 x

*DNS 네임스페이스
:도메인 주소를 어떻게 관리할지에 대한 방법

<Route 53>

6) AWS Step Functions

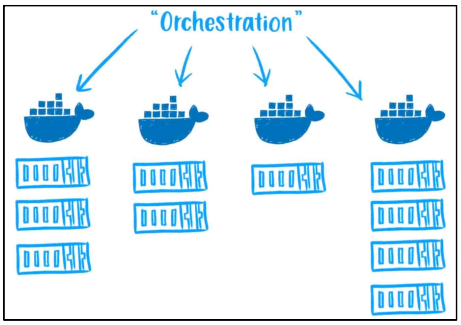
-서버리스 워크플로를 시각적으로 구성할 수 있는 기능

-주로 람다 함수를 오케스트레이션하는데 활용

오케스트레이션이란 무엇입니까?

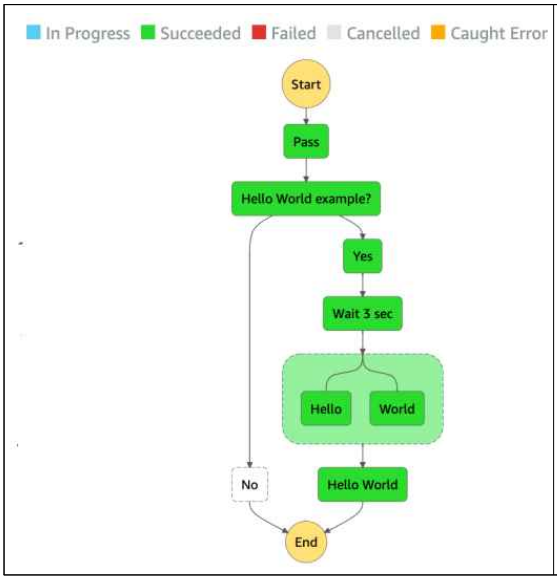
오케스트레이션은 여러 개의 컴퓨터 시스템, 애플리케이션 및/또는 서비스를 조율하고 관리하는 것으로, 여러 개의 작업을 함께 연결하여 크기가 큰 워크플로나 프로세스를 실행하는 방식을 취합니다. 이러한 프로세스는 여러 개의 자동화된 작업으로 구성될 수 있으며 관련되는 시스템도 여러 개일 수 있습니다.

오케스트레이션의 목표는 빈도가 높고 반복할 수 있는 프로세스의 실행을 간소화 및 최적화하여 데이터 팀이 복잡한 작업과 워크플로를 간편하게 관리하도록 돕는 것입니다. 프로세스를 반복할 수 있고 작업을 자동화할 수 있다면 오케스트레이션을 사용하여 시간을 절약하고 효율성을 증대하고 중복성을 없앨 수 있습니다. 예를 들면, 작업 오케스트레이션을 통해 데이터 및 머신 러닝을 간소화할 수 있습니다.



-각 그래프 단계별로 해당 단계의 결과에 따라 다음으로 수행하는 작업이 무엇인지 정의함

-좀 복잡한 워크플로 만들어 AWS에서 실행시킬 수 있는 편리한 도구



- 제공하는 기능
  :시퀀싱, 병행 실행, 조건 설정, 타임 아웃, 에러 처리하기 등
- 람다 뿐만이 아니라
  EC2, ECS, On-premises 서버, API Gateway, SQS 큐 등
  다양한 AWS 서비스를 워크플로우에 넣을 수 있음
- 워크플로에 사람이 개입해서 승인을 해야만 진행되는 단계 설정 가능
  ex) 어떤 지점에 이르러서는 사람이 결과를 확인하고 승인해야 다음 단계로 넘어감
- Use cases: 주문 이행, 데이터 처리, 웹 app 구성,
  복잡한 워크플로를 시각적으로 구성하려고 할 때