AWS Summit (2023.05.04)

작성자 : 이지현

⏰ Time Table

가격은 저렴, 성능은 최대로! 확 달라진 Amazon EC2 알아보기 **L100 - L200**	"이봐, 해봤어?" 해본! 사람의 Modern Data Architecture 비밀 노트 **L100 - L200**	잘나가는 애플리케이션 성능? 알맞은 스토리지로부터! **L200**	오픈소스 데이터베이스로 탈 오라클! Why not? **L200**	Amazon EKS, 중요한 건 꺾이지 않는 안정성 **L200**	바쁘다 바빠, 현대사회! Amazon Kendra로 원하는 자료를 적재적소에 찾아서 활용하기 **L200**	AWS에서 OpenTelemetry 기반의 애플리케이션 Observability 구축/활용하기 **L300**	스타트업의 서버리스 기반 SaaS 데이터 처리 및 데이터 웨어하우스 구축 사례
스마트한 클라우드 스토리지 비용 관리 전략 **L100 - L200**	AWS의 개발자를 위한 신규 서비스 소개 Amazon CodeCatalyst & Amazon CodeWhisperer **L100 - L200**	AWS의 관리형 VDI 서비스! 알고 계셨나요? **L200**	데이터 공유와 활용! 전사 데이터 거버넌스를 위한 3가지 비밀 무기 **L200**	천만 사용자 서비스를 위한 Amazon SageMaker 활용 방법 진화하기 **L200**	기업 고객 대상 기계학습 기반 콜센터 도입을 위한 여정 **L300**	갤럭시 규모의 서비스를 위한 Amazon DynamoDB의 역할과 비용 최적화 방법 **L300**	Amazon EKS 데이터 전송 비용 절감 및 카오스 엔지니어링 적용 사례
성공적인 AWS RDS 마이그레이션을 위한 여정과 필수 고려사항 **L200**	AWS 와 Cisco가 함께하는 제로 트러스트 **L100**	당신만 모르고 있는 AWS 컨트롤 타워 트렌드 **L100**	Confluent와 함께하는 실시간 데이터와 클라우드 여정 **L200**	통합을 통한 보안 간소화 **L100**	하시코프가 제안하는 클라우드 배포/운영/보안강화 전략 **L100**	클라우드 낭비 감소를 위한 더 많은 최적화 **L100**	OTT 서비스는 어떻게 콘텐츠를 보호하는가 - 콘텐츠 보안 기술의 동향과 Media Package를 통한 다중 키 암호화 **L200**
지속적인 혁신과 발전, AWS 네트워킹이 이끄는 미래 **L100 - L200**	비즈니스 경쟁에서 승리하기 위한 AWS AI/ML 서비스 **L200**	AWS에서 최소한의 비용으로 구현하는 멀티리전 DR 자동화 구성	서버리스, 이제는 데이터 분석에서 활용해요! **L300**	아마존의 공급망 전략을 배워보고, 우리 회사에 적용하기 **L100 - L200**	다중 계정 및 하이브리드 환경에서 안전한 IAM 체계 만들기 **L200**	12가지 디자인 패턴으로 알아보는 클라우드 네이티브 마이크로서비스 아키텍처	Amazon Neptune 및 Elastic을 이용한 추천 서비스 및 검색 플랫폼 구축하기
MongoDB Atlas와 함께하는 Developer Data Platform **L100**	Datadog을 활용한 AWS 서버리스 Observability **L200**	Snowflake: 모든 데이터 워크로드를 위한 하나의 클라우드 데이터 플랫폼 **L100**	데이터, 분석 및 AI를 통합하는 단 하나의 레이크하우스, Databricks on AWS 로 시작하기 **L100**	클라우드 보안의 새로운 접근법 **L200**	SaaS와 모니터링 플랫폼 **L200**	KINX CloudHub를 통한 AWS Direct Connect 연동: 네트워크 확장 방안 **L200**	글로벌 대화형 서비스 개발 플랫폼 Sendbird가 AWS와 함께한 빌드 여정 **L100**
지능화되는 랜섬웨어 위협으로부터 지킬 것인가? 당할 것인가? **L100 - L200**	클라우드 솔루션 비즈니스를 위한 게임 체인저: AWS Marketplace **L100 - L200**	AWS Graviton과 함께하는 계획문제 최적화 애플리케이션 개발 **L200**	모두를 위한 BI, QuickSight **L200**	진짜로 코드 없이 기계학습 모델을 만드는 것이 가능하다구요? SageMaker로 No/Low 코드 기계학습 해 보기	클라우드의 경계를 허무는 AWS Hybrid Cloud Services **L300**	실시간 CDC 데이터 처리! Modern Transactional Data Lake 구축하기 **L300**	생성 AI 모델의 임베딩 벡터를 이용한 서버리스 추천 검색 구현하기 **L300**

Modern Data Arch.

모던 데이터 아키텍처란?

- 다양한 저장소의 데이터를 접근제약 없이 분석
- 분석 및 기계 학습 서비스에 손쉽게 연결
- 데이터 액세스, 보안, 거버넌스 통합관리
- 저렴한 비용, 최고의 성능, 확장 가능한 아키텍처

개요

- 데이터 소스 ➡ (수집 ➡ 처리 ➡ 사용) ➡ 사람
- 효율적인 의사결정 / 예측 및 최적화 / 새로운 기회 발굴 을 위해 데이터를 분석함
- 다양한 데이터 저장소에 대한 접근은 각각임
  - 비효율 / 시간 / 비용 등의 발생
- 저장소를 연결한 "확장 가능한 데이터 레이크"
  - 심지어 다른 AWS 계정까지도…
  - dev / test / prod 까지도…
  - 다양한 형태의 DB 와의 연결 + 분석 도구 와의 연결이 가능함
  - 그런 형태들에 대한 각각의 권한과 인증을 관리해야함 ➡ (하나의 통합 된 관리자/생산자/소비자) 를 관리할 수 있음
- 높은 비용, 성능과 확장성의 한계가 존재함
- AWS Glue (ETL, 데이터 카탈로그)

사례

🔨 nepes

- 서버리스, AWS service 만으로 구성
- 데일리 데이터 스크랩 ➡ 하루에 4차례 ➡ 실시간
- 한장의 이미지에 대한 불량 이미지 분석
- + 25 공정에 대한 이미지 분석
- + 다른 데이터까지 통합해 불량 분석

⚠️ 단계적으로 진행

🔨 encored

- 발전량 예측 관련 비즈니스
- 이를 위해서, 외부 데이터까지의 결합이 필요함
- Kafka (대용량) + lambda(소량) 로 수집

⚠️ 수단과 목적을 헷갈리지 말자

⚠️ 비지니스를 수행하기 위해서 레이크를 구성하고 분석을 하는 것!

🔨 드라마앤컴퍼니

- 데이터 샌드박스 구축을 통해, 분석가가 직접 마트를 구축할 수 있도록 구성
- 이기종 데이터를 어떻게 통합하는지가 관건

⚠️ 데이터통합 100%는 안되겠지만, 데이터 카탈로그는 통합관리가 충분히 가능하다!

🔨 푸드테크

- AWS 서비스를 사용해, 기술 재교육 시간 최소화
- 오픈 소스 기반으로 신규 팀원의 소프트 랜딩 가능
- 기술 적응 기간이 줄어든다!

RDS Migration

RDS , Aurora

- Aurora : io 로 과금체계

Migration Step

1. **Assess** : 분석 인프라 환경, 요구조건, 리스크 선제 분석
    - DB Server : 리소스 사용량의 패턴 등을 확인
    - DBMS : version, license, option etc..
    - Business : Read/Write ratio, connected application feature
2. Plan : 용량, 기관, 다운 타임, 롤백계획 등 계획 수립
3. Test : 시간측정 포함 (리허설 개념)
4. Execute : 실행, 검증, 오픈, 모니터링

고려사항

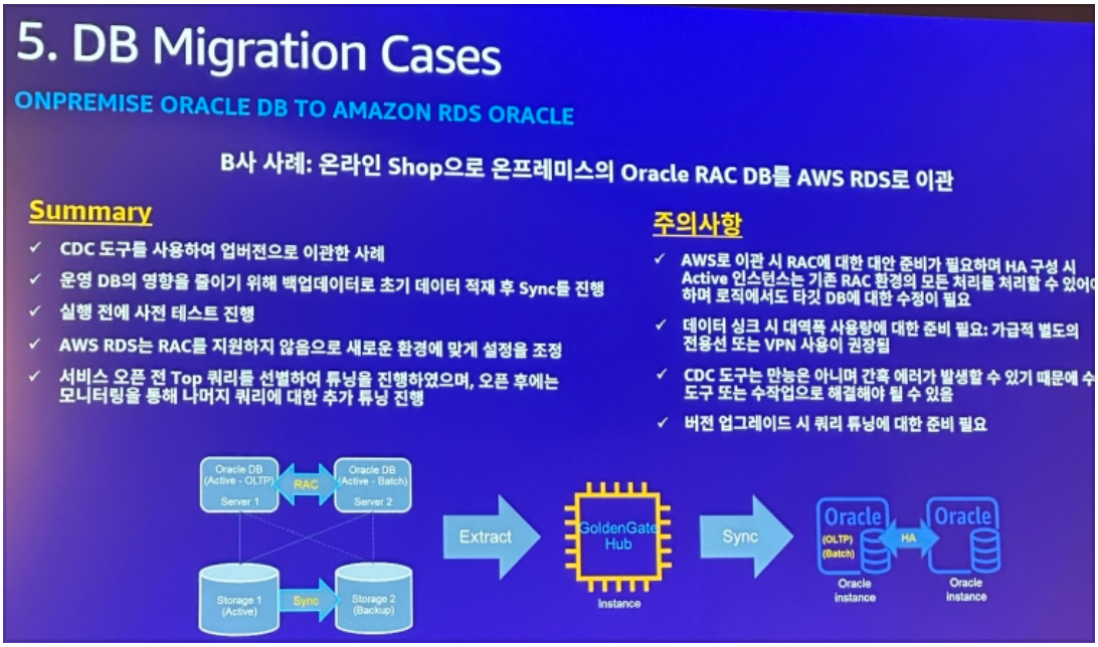
- ❗ **백업** 리스토어 에러를 위한 2벌 이상
- 호환성 점검
- 이관방법 및 제약사항 숙지
- 라이센스 준비
- 3rd party SW 이관
- 충분한 시간 확보

Migation Tools

1. OS Commands
    - 간단한 OS 명령만을 이용
    - 파일복사에 따른 중단시간이 용량에 비례함
    - OS, 버전 등의 환경이 일치해야함
    - 복사 중 끊길 수 있음
2. DBMS Provided Tools

    - DB version 의 호환, 테이블 선별 이전 가능
    - 시간이 많이 소요됨
3. CDC
    - OS 등의 제약이 없음
    - 타 DB, 대용량 가능
    - On going 수준의 복제가 가능함
    - 비용 발생 & 지원이 안되는 DB가 있음

예제

1. MS SQL ➡ RDS mySQL
    - 이기종이라 제약사항 점검 필요
    - 인덱스, 키 등의 제약
    - DDL 이름 등
    - 쿼리 및 로직 수정 다수 발생
2. on-premise ORACLE RAC ➡ AWS RDS ORACLE DB
    - 

DR in AWS

복구시점 및 복구시간 목표

복구시점목표 (Recovery Point Object / RPO) : 재해발생할 경우 데이터 손실 구간

복구시간목표 (Recovery Time Object / RTO)

1. Backup & Restore : 리소스 전체 재생성 & 비용저렴
2. Pilot light : 이벤트 발생 후에 리소스 생성 및 확장
3. Warm Standby : 서비스 가능한 수준으로 상시 구동
4. Multi-site activate : 무중단 & 비쌈….

- 비용과 서비스 중단 기간은 Trade Off
- 수용가능비용과 복구 목표시간을 고려해 방식 선택

멀티리전 재해복구

- AWS Backup (Backup & Restore)
  - 정기적인 백업
  - 데이터 백업
  - **메타 데이터 백업** (vpc config ➡ CMDB, AWS API, IaC (Infrastructure as Code)
  - DR Region 에 Backup Vault 구성 및 이벤트 발생시 Restore
  - AWS Backup은 각기 다른 자원의 백업 방식을 자동화 통합함

IaC 관리 및 백업

- 버전 관리 필요 (git)
- **IaC 코드내 의존성 분리**
  1. region 별로 의존성이 강한 코드를 분리해야, 백업 리전에서도 정상작동이 가능함)
  2. 동적인 정보는 자동적으로 관리 될 수 있도록 구성
  3. 관리형 리소스도 추상화해서 간접접근토록 구성
- 복구는 DR Region 의 리소스만을 이용해서 하도록!
- 지속적인 검증 및 보완

AWS Elastic Disaster Recovery (DRS) (Pilot light)

- 모든 소스에서 복제
- 다양한 OS, DB 등 지원
- prod 에 영향없는 DR 훈련 가능
- 온프레미스 ➡ AWS / 타 CSP ➡ AWS / Region, 가용영역 간 모두 가능

1. Set up : 지속적인 복제 시작 (prod 에 AWS Replication Agent 설치 필요)
2. Test : 무중단 테스트를 위한 인스턴스 시작
3. Operate : 모니터링 및 훈련 가능
4. FailOver : 실제 재해 발생시
5. 필요시 Failback

Serverless Observability with Datadog

Datadog

- 모니터링 BI 및 바운드, 통계 등
- 다양한 리소스 및 플랫폼 메트릭 수집
- 로그 중앙화

- 다양한 언어의 end to end 분산 트레이싱 (트랜잭션을 연결해서 모니터링)
- 머신러닝 기반 분석 및 알림
- 유연한 health check (단순 헬스체크가 아닌, 점검 수준까지 자동화)
- 사용자 화면까지 로깅떠버림….(우와!!!) 😬
- workflows 구성으로, 스크립트를 슬랙 등으로 처리가 가능….(우와!!!) 😬

Serverless

- 람다의 설계가 잘못될 경우, 비용상승과 직결됨
- Observability 통해서 개선점 파악(?)
- invoke 메트릭까지 포함해, 트랜잭션을 분석해 불필요한 라이브러리 등 리소스 제거로 latency, 비용 등 개선 가능

CDC

- RDBMS - Scalability 한계 - Read 부하는 어떻게 되더라도, Write 부하는 확장성 한계가 존재함
- Distributed File System - 이러한 한계를 극복가능함
- stream storage, delivery 등의 data pipe line 설계해서 Data Lake(S3) 구성
- RDBMS 의 트랜잭션 기능을 S3 에서는 사용 불가능함…
    - 트랜잭션 RDBMS CDC 데이터를 어떻게 S3 에 담을 수 있을까?!

1. S3 에서 Inserted Table + Update, Delete Table 따로 구성
2. 테이블을 합쳐서 확인하는 View table 구성
3. logical view (조회시점에 쿼리 조합) vs Materialized View (별도로 view 테이블 object 보관 - AWS Redshift)

logical view 단점

- 히스토리가 많은 데이터는 read 시점에 부하가 생길 수 있음 ➜ (해결책?) 일정 시점마다 연산을 해놓는다…
- 과거 데이터를 따로 보관할 경우, logical view 사용할 때 불러와서 조합이 어려워 진다…(데이터 소스가 다른 곳에 있으니까!)

Materialized view 단점

- 무한히 큰 저장소를 들고 있을 수 없음

S3 에서 Materialized view 구현하기

- Table Format (= Layout of Files in Table)
- commit log 를 쌓고,
- 참고해서 indexing 해서 데이터를 조합해서 보여주기
- hudi, iceburg, delta lake

- file 과 관련된 & 변경사항에 대한 meta data 와 실제 변경 data 를 따로 저장
- commit log 를 관리하기 때문에, Time Travel 가능