

데이터 레이크 활용 실습

AWS Lake Formation Workshop 실습

데이터 레이크 활용 실습과 자동화

1. 데이터 레이크 개요 및 아키텍처

1.1 데이터 레이크란 무엇인가

데이터 레이크(Data Lake)는 원시 상태의 다양한 형식의 데이터를 저장할 수 있는 중앙 저장소이다. 구조화된 데이터(예: RDBMS), 반구조화 데이터(예: JSON, CSV), 비정형 데이터(예: 이미지, 오디오) 등 모든 유형의 데이터를 저장할 수 있다.

데이터는 수집 즉시 저장되며, 처리나 분석은 필요한 시점에 수행된다. 이로 인해 유연성과 확장성이 뛰어나며, 다양한 분석 및 머신러닝 작업에 적합하다.

1.2 전통적 DW와의 차이점

전통적 데이터 웨어하우스(DW)는 스키마가 고정된 구조화 데이터를 대상으로 하며, 스키마를 저장 시점에 정의(schema-on-write)해야 한다. 반면 데이터 레이크는 schema-on-read 방식으로 저장하고, 분석 시점에 스키마를 정의한다.

항목	데이터 웨어하우스	데이터 레이크
데이터 형식	구조화	모든 형식 지원
스키마 적용	저장 시점	조회 시점
저장소	RDB 기반	객체 스토리지(S3 등)
주요 사용 사례	BI 리포트, 회계 분석	로그 분석, ML, IoT 등

1.3 서버리스 아키텍처 개념

서버리스(Serverless)는 사용자가 서버를 프로비저닝하거나 관리할 필요 없이 코드 실행, 데이터 처리, API 제공 등이 가능한 구조이다. AWS Lambda, Glue, Athena, EventBridge 등이 이에 해당하며, 비용 효율성과 탄력성이 뛰어나다.

서버리스 데이터 레이크는 다음과 같은 특징을 가진다:

- 인프라 유지보수 불필요
- 필요할 때 자동 확장
- 이벤트 기반 처리 가능
- 사용한 만큼 과금

1.4 AWS 기반 데이터 레이크 주요 컴포넌트

- **Amazon S3:** 모든 데이터 저장소 역할, 계층 구조 지원
- **AWS Glue:** 크롤러, ETL, 데이터 카탈로그를 포함한 데이터 처리 플랫폼
- **Amazon Athena:** SQL 기반 쿼리 서비스. 서버리스이며 S3 데이터를 직접 조회 가능
- **Lake Formation:** 데이터 레이크의 보안 및 권한 관리 통합 서비스
- **Amazon EMR:** Apache Spark, Hive, HBase 등 빅데이터 처리를 위한 클러스터 플랫폼

1.5 데이터옵스 개요

DataOps는 데이터 파이프라인의 **신뢰성, 자동화, 품질**을 개선하기 위한 접근법이다. DevOps의 원칙을 데이터 분석과 처리에 적용한 개념으로 다음을 포함한다:

- 데이터 품질 테스트 자동화
- 파이프라인 코드의 Git 기반 버전관리
- CI/CD 기반 배포 자동화
- 모니터링 및 자동 롤백 기능

2. SDLF 기반 데이터 레이크 구축 실습

2.1 초기 환경 설정

2.1.1 프로젝트 구조 이해

SDLF(Serverless Data Lake Framework)는 AWS에서 제공하는 오픈소스 템플릿으로, 다음과 같은 계층으로 구성된다:

- raw: 수집한 원시 데이터 저장
- curated: 정제 및 필터링된 데이터
- trusted: 분석/ML 등에 직접 사용 가능한 검증 데이터

각 계층은 S3에 버킷/폴더로 구분되며, 처리 파이프라인은 Step Functions, Glue, Lambda를 통해 자동화된다.

2.1.2 도메인 및 계층 구조 설계

- 도메인: 기능 또는 부서 기준으로 구분 (예: marketing, finance)
- 계층: 데이터 품질과 처리 단계에 따른 구분 (raw → curated → trusted)

2.1.3 IAM 역할 구성

- SDLFExecutionRole: Step Function 실행 권한
- GlueCrawlerRole: S3에 접근하여 메타데이터 수집 권한
- LambdaExecutionRole: S3, Glue, EventBridge와 상호작용할 수 있는 권한

2.2 스토리지 계층 배포 (S3 기반 레이크)

2.2.1 S3 버킷 생성 및 계층화

S3 버킷은 도메인 및 계층 구조에 따라 계층화한다. 예:

- s3://datalake/marketing/raw/
- s3://datalake/finance/trusted/

2.2.2 S3 버킷 정책 및 권한 구성

- 정책 예:

```
{  
  "Effect": "Allow",  
  "Action": ["s3:GetObject", "s3:PutObject"],  
  "Resource": "arn:aws:s3:::datalake/*"  
}
```

- 버킷 ACL 및 SSE 암호화 설정

2.3 데이터 카탈로그화

2.3.1 Glue Crawler 설정

- Crawler를 생성하여 s3://datalake/raw/를 대상으로 스캔
- 테이블명 자동 지정: domain_layer_table

2.3.2 메타데이터 자동 수집

- 컬럼명, 데이터 형식, 파티션 구조 자동 인식
- 크롤 주기 설정: 1시간, 하루, 수동 등

2.3.3 Athena에서 구조 확인

- SELECT * FROM marketing_raw_logs LIMIT 10;

- 테이블 미리보기, 쿼리 실행, 결과 S3 저장

2.4 데이터 처리 (ETL/ELT)

2.4.1 Glue Spark Job 구성

- PySpark 코드 예:

```
from awsglue.context import GlueContext
from pyspark.context import SparkContext
```

```
glueContext = GlueContext(SparkContext())
```

```
df = glueContext.create_dynamic_frame.from_catalog(database="datalake",
table_name="marketing_raw")
```

```
filtered = df.filter(lambda x: x["status"] == "success")
```

```
glueContext.write_dynamic_frame.from_options(frame=filtered, connection_type="s3",
connection_options={"path": "s3://datalake/marketing/curated/"}, format="parquet")
```

2.4.2 Lambda를 이용한 실시간 처리

- S3에 파일 업로드 이벤트 발생 → Lambda 트리거 → Step Function 시작
- Lambda가 Glue Crawler 실행 요청 또는 메타데이터 등록 수행

2.4.3 파티셔닝 및 포맷 전환

- Parquet, ORC 등 컬럼 기반 포맷으로 저장
- partitionKeys=["year", "month", "day"] 적용

2.5 데이터 소비

2.5.1 Athena 기반 SQL 분석 실습

- SELECT COUNT(*) FROM curated.marketing_logs WHERE campaign='X';

2.5.2 S3 Select, Redshift Spectrum

- S3 Select: 대용량 객체 내 일부 필드만 조회 가능
- Spectrum: Redshift에서 외부 테이블로 직접 쿼리

2.5.3 API/BI 도구 연동

- QuickSight에서 Glue Catalog 테이블을 데이터 소스로 등록
- REST API로 Athena 결과 반환

2.6 정리

2.6.1 환경 구성 점검

- IAM 정책 유효성, Step Function 상태, Lambda 로그 확인

2.6.2 모듈화/템플릿 정비

- CDK 또는 CloudFormation 템플릿으로 인프라 구성 모듈화

2.6.3 리소스 정리 자동화

- 사용 종료 시 Glue Job, S3 객체, 로그 그룹 자동 정리
-

3. 데이터 레이크 운영 자동화와 DataOps 도입

3.1 다중 도메인 및 환경 전략

3.1.1 개발/운영 환경 분리 전략

- dev, stage, prod 환경으로 나누어 실험과 운영을 분리

- 예: datalake-dev, datalake-prod S3 버킷 분리
- Step Function, Lambda, Glue Job 등도 환경별 별도 설정

3.1.2 모듈리식 vs 도메인 기반 처리

- 모듈리식 구조는 모든 처리를 단일 파이프라인에서 수행 → 장애에 취약
- 도메인 기반 구조는 각 업무 도메인마다 독립된 파이프라인 구성 → 장애 격리, 유지보수 편리
- SDLF는 도메인 별 파티셔닝 및 모듈화를 통해 도메인 중심 설계를 지원함

3.2 CI/CD 파이프라인 구성

3.2.1 CodePipeline + CodeBuild 구성 이해

- CodeCommit (소스) → CodeBuild (빌드/배포 스크립트) → CodePipeline (워크플로우)
- 각 파이프라인은 Glue 스크립트, Lambda 함수, Step Function 정의 등을 자동 배포

3.2.2 기초 CICD: 기본 배포 파이프라인 구축

- 템플릿 위치: cicd/base-cicd.yaml
- CodePipeline을 통해 SDLF 핵심 인프라 자동 생성
- 구성 요소: S3, Glue, IAM, Lambda, Step Functions 등

3.2.3 관리팀 CICD: 관리 도메인 중심 배포 전략

- 운영 팀이 관리하는 공통 리소스(Catalog, IAM, 알림 시스템 등)를 별도로 배포
- 소스 코드 예: infra/shared-infra/*
- 리소스 변경 시 팀 승인 및 자동화 테스트 적용

3.2.4 SDLF 팀 CICD: 파이프라인 재사용 및 협업 구조

- 도메인별로 정의된 재사용 가능한 구성 모듈 활용 (ex. Glue Job 템플릿)
- CodePipeline 브랜치별 배포 환경 설정 (dev, main)
- PR 승인 시 자동 배포 트리거 가능

3.3 자동화 운영과 이벤트 기반 처리

3.3.1 EventBridge, SNS, Lambda 활용

- S3 업로드 이벤트 → EventBridge 룰 → Lambda 트리거
- 처리 결과를 SNS로 알림 전송 (예: Slack, Email)
- 이벤트 예: Glue Job 실패 → 알림 발송 + 자동 재시도

3.3.2 오류 알림 및 자동 롤백 처리

- Step Function에서 실패 상태 감지 시 SQS 또는 SNS로 알림 전송
- Lambda가 이전 상태를 백업 → 롤백 실행
- CloudWatch Alarms + Auto Disable 구조 적용 가능

3.3.3 S3 이벤트 기반 파이프라인 트리거

- s3:ObjectCreated:* → Lambda → Glue Crawler → ETL → 카탈로그 반영
 - 완전한 서버리스 흐름 구성 가능
-

4. 운영 분석 및 시각화 실습

4.1 CloudWatch 로그 및 메트릭 수집

- Lambda, Glue, Step Function 모두 CloudWatch에 로그 저장

- 예: Lambda 로그 스트림 검색 → 오류 확인
- CloudWatch Logs Insight 쿼리 예:

fields @timestamp, @message

| filter @message like /Error/

| sort @timestamp desc

4.2 Athena 로그 분석 쿼리 작성

- CloudTrail, CloudWatch 로그를 S3에 저장 후 Athena에서 분석
- 예: 실패한 Step Function 실행 목록 조회

SELECT * FROM cloudtrail_logs

WHERE eventName = 'StartExecution'

AND errorCode IS NOT NULL;

4.3 QuickSight 대시보드 생성 실습

- Athena 테이블을 QuickSight 데이터셋으로 등록
- 분석 지표 예: 일별 처리 건수, 실패율, 평균 처리 시간
- 시각화 유형: 막대 그래프, KPI 카드, 파이 차트

4.4 운영 대시보드 자동 배포

- QuickSight 템플릿 활용한 대시보드 배포
- CloudFormation 또는 boto3 API 통해 자동화 가능

5. 실전 적용 및 마무리

5.1 구축된 데이터 레이크 검토

- 각 도메인 → ingestion → curation → consumption 계층 흐름 점검
- IAM 정책, Glue 카탈로그, S3 구조 재확인

5.2 비즈니스 시나리오 적용 예

- 마케팅 로그 분석: 광고별 클릭/전환 분석, A/B 테스트 로그 분석
- 제조 설비 모니터링: 센서 데이터 → 실시간 처리 → 알림 + 리포트 생성

5.3 비용 최적화 포인트

- S3 Intelligent-Tiering 적용으로 비활성 데이터 자동 비용 절감
- Glue Job 시간 단축 (필터링, 파티셔닝 적용)
- Athena 쿼리 최적화: SELECT column1 vs SELECT *

5.4 향후 확장 방향

- SageMaker 연동하여 ML 예측 파이프라인 구현
 - Kafka → Firehose → S3 실시간 수집 아키텍처 확장
 - Redshift, Lake House 아키텍처와 통합 가능성
-

CI/CD (지속적 통합 및 지속적 배포) 파이프라인은 DevOps 원칙에 기반한 자동화된 애플리케이션 개발 및 배포 프로세스를 구현하는 데 사용된다..

✅ 1. CI/CD란 무엇인가?

- **CI (Continuous Integration)**

개발자가 자주 코드를 리포지토리에 병합하고, 자동으로 빌드 및 테스트가 수행되도록 구성하는 것.

- **CD (Continuous Delivery / Deployment)**

빌드된 애플리케이션을 자동으로 대상 환경에 배포하거나, 프로덕션까지 푸시하는 것.

- *Delivery*: 수동 승인 후 배포
- *Deployment*: 자동 프로덕션 배포

✅ 2. AWS에서의 CI/CD 핵심 서비스 구성

서비스	역할
AWS CodeCommit	Git 기반의 소스 코드 저장소
AWS CodeBuild	코드 빌드 및 테스트 수행
AWS CodePipeline	전체 CI/CD 워크플로우 정의
AWS CodeDeploy	EC2, Lambda, ECS 등 대상에 애플리케이션 배포
AWS CloudFormation / CDK	인프라 코드(IaC)를 통해 배포 리소스를 정의
Amazon S3	아티팩트, 정적 파일, 배포용 ZIP 저장
AWS Lambda	커스텀 작업 수행(예: 알림, 검증 등)

✅ 3. 전체 파이프라인 흐름 예시

[CodeCommit] → [CodeBuild] → [Test 단계] → [Approval(Optional)] → [CodeDeploy] 또는 [Lambda 배포]

✅ 4. 실습 예시: 서버리스 앱 CI/CD 구성 (Lambda + S3 + CodePipeline)

4.1 리포지토리 구성

/lambda-app/

```
|—— handler.py
|—— requirements.txt
|—— buildspec.yml
```

4.2 buildspec.yml 예시 (CodeBuild에서 참조)

version: 0.2

phases:

install:

runtime-versions:

python: 3.9

commands:

- pip install -r requirements.txt -t .

build:

commands:

- zip -r lambda.zip .

artifacts:

files:

- lambda.zip

4.3 CodePipeline 구성 단계

1. **Source:** CodeCommit 저장소 또는 GitHub 연결
 2. **Build:** CodeBuild 프로젝트 실행 → lambda.zip 생성
 3. **Deploy:** Lambda 함수에 zip 파일 자동 배포
-

✅ 5. 다양한 배포 유형

배포 대상	방법
EC2	CodeDeploy + EC2 에이전트 사용
Lambda	CodePipeline → Lambda 직접 연결
ECS	CodeDeploy 또는 blue/green 방식 지원
S3 (정적 웹사이트)	CodePipeline이 aws s3 sync 사용
Amplify	Git 연동만으로 자동 배포 가능 (프론트엔드 전용)

✅ 6. 고급 기능 및 확장성

6.1 Approval 단계 추가

- 수동 승인 단계 삽입 (ex. 운영자 검토 후 배포)

6.2 CloudWatch 이벤트와 연동

- 배포 실패 시 알림 또는 롤백 시도 가능

6.3 Slack / Email 알림

- SNS, Chatbot 연동으로 배포 성공/실패 메시지 전달

6.4 Canary / Blue-Green 배포

- CodeDeploy에서 트래픽 점진 분산으로 무중단 배포 구현

✅ 7. DataOps와의 연계 (데이터 파이프라인 CI/CD)

데이터 프로젝트에서도 CI/CD가 가능함. 예:

- Glue Job PySpark 코드 → CodeBuild로 테스트 → S3에 배포
- Step Function 정의 → CloudFormation 자동 배포
- Athena SQL / Lake Formation 권한 → Git 관리 + 배포

✅ 8. 실습용 CloudFormation 예제 (CI/CD 파이프라인 생성)

Resources:

MyPipeline:

Type: AWS::CodePipeline::Pipeline

Properties:

RoleArn: arn:aws:iam::123456789012:role/CodePipelineRole

Stages:

- Name: Source

Actions:

- Name: SourceAction

ActionTypeId:

Category: Source

Owner: AWS

Provider: CodeCommit

Version: 1

Configuration:

RepositoryName: lambda-app

BranchName: main

OutputArtifacts:

- Name: SourceOutput

- Name: Build

Actions:

- Name: BuildAction

ActionTypeId:

Category: Build

Owner: AWS

Provider: CodeBuild

Version: 1

Configuration:

ProjectName: LambdaBuild

InputArtifacts:

- Name: SourceOutput

OutputArtifacts:

- Name: BuildOutput

✓ 9. 요약

- AWS에서는 풀매니지드 CI/CD 서비스를 통해 개발 → 테스트 → 배포 과정을 자동화할 수 있다.
- CodeCommit, CodeBuild, CodePipeline, CodeDeploy가 핵심 서비스이며, 다른 AWS 서비스와도 쉽게 연동된다.
- Lambda, ECS, S3, EC2, Glue 등 다양한 대상에 배포가 가능하며, 운영 효율성과 코드 품질을 동시에 향상시킬 수 있다.

The screenshot shows the AWS Workshop Studio interface. The left sidebar contains a navigation menu with the following items: '서버리스 데이터 레이크 프레임워크 워크숍' (Serverless Data Lake Framework Workshop), 'SDLF 배포' (SDLF Deploy), '초기 설정' (Initial Setup), '스토리지 계층 배포' (Storage Layer Deploy), '데이터 카탈로그화' (Data Cataloging), '데이터 처리' (Data Processing), '데이터 소비' (Data Consumption), '정리하다' (Organize), '생산을 치다' (Production), '다중 도메인 및 환경 전략' (Multi-domain and Environment Strategy), '기초 CI/CD' (Basic CI/CD), '관리된 CI/CD' (Managed CI/CD), 'SDLF 팀 CI/CD' (SDLF Team CI/CD), '더 나아가기' (Go Further), '축하해요!' (Congratulations!), '정리하다' (Organize), '아카이브: CodeCommit을 사용한 SDLF' (Archive: SDLF using CodeCommit), and '아카이브: SDLF 1.0' (Archive: SDLF 1.0). The main content area is titled 'Serverless Data Lake Framework Workshop' and '서버리스 데이터 레이크 프레임워크 워크숍'. It includes a welcome message, a paragraph about the benefits of a data lake, a paragraph about SDLF, and a link to the workshop materials. At the bottom, there are logos for Naranja Finance - Argentina and Formula 1 Motorsports - United Kingdom.

aws workshop studio

서버리스 데이터 레이크 프레임워크 워크숍

Serverless Data Lake Framework Workshop

서버리스 데이터 레이크 프레임워크 워크숍

Serverless Data Lake Framework(SDLF) 워크숍에 참여해 주셔서 감사합니다!


데이터 레이크는 조직에 민첩성을 제공합니다. 소비자가 필요한 데이터를 빠르게 찾아 비즈니스 프로젝트에 사용할 수 있는 수 있습니다. 파일을 저장하는 것 외에도 고려해야 할 사항이 많습니다. 예를 들어, 저장된 내용을 알 수 있도록 데이터를 카탈로그화? 데이터 품질을 어떻게 관리합니까? 변환을 위한 코드를 소스 제어 하에 유지하는 방법은 무엇입니까? 개발, 테스트 및 프로덕션을 구축하는 데는 몇 주가 걸릴 수 있으며 이 시간을 데이터 혁신과 비즈니스 목표 달성에 사용하는 것이 좋습니다.


SDLF는 AWS에서 엔터프라이즈 데이터 레이크의 제공을 가속화하고 프로덕션까지의 배포 시간을 몇 달에서 몇 주로 단축하는 파트너 및 고객이 모범 사례에 따라 데이터 레이크의 기본 구조를 구현하는 데 사용할 수 있습니다.

이 워크숍에서 논의된 다양한 구성 요소와 지원 저장소는 [설명서](#)에 자세히 나와 있습니다. [\[?\]](#)

[소스 코드](#) [문의하기](#) [선적 서류 비치](#)

공개 참조:


Naranja
Finance - Argentina

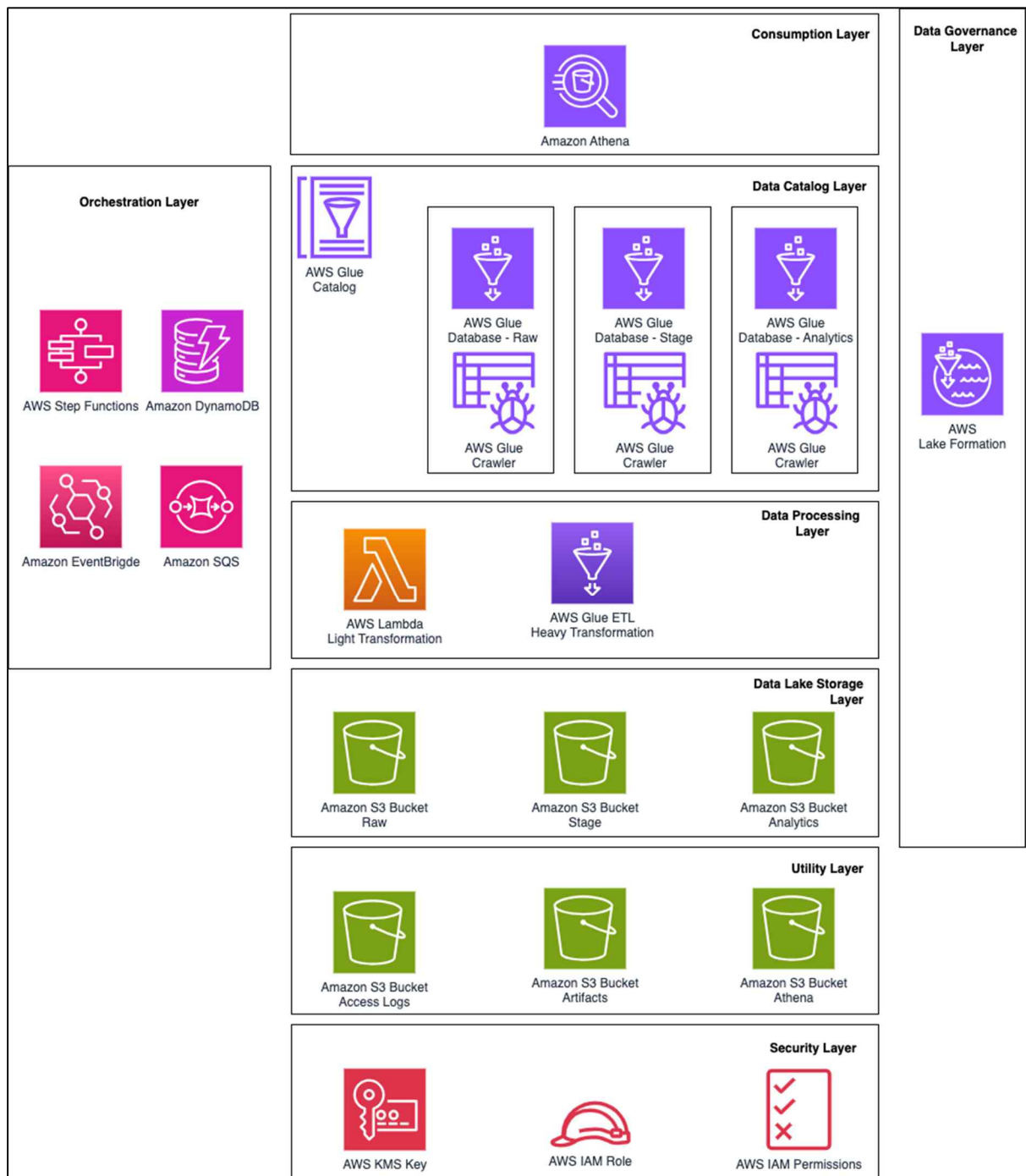

FORMULA 1
Motorsports - United Kingdom

[실습 시작]

아래 Workshop 실습을 수행한다

<https://catalog.us-east-1.prod.workshops.aws/workshops/501cb14c-91b3-455c-a2a9-d0a21ce68114/en-US>

[서버리스 데이터 레이크 프레임워크 전체 구조]



층	설명
저장	S3 및 Lake Formation을 사용한 데이터 레이크 스토리지 계층
목록	Glue 데이터 카탈로그(데이터베이스 및 크롤러)
처리 중	EventBridge 트리거 Lambda 함수 및 Glue 작업으로 데이터 처리 시연
소비	데이터 소비를 위한 Athena 작업 그룹
관현악법	Step Functions 및 EventBridge
거버넌스와 보안	LakeFormation, KMS 키 및 IAM 역할

초기 설정

배포 단계

AWS CloudShell을 열고 터미널에서 아래 명령을 실행

```
curl -L -O https://github.com/aws-labs/aws-serverless-data-lake-
framework/archive/refs/tags/2.10.0.tar.gz
tar xzf 2.10.0.tar.gz
cd ./data-lakes-on-aws-2.10.0/
```

CloudFormation 템플릿을 준비할 두 개의 CodeBuild 프로젝트를 만듭니다.

```
cd sdlf-cicd/
./deploy-cicd.sh workshop
```

(이후 실습은 Workshop 문서 참조)

. 위의 CodeBuild 프로젝트에서 데이터 레이크 인프라를 배포하는 데 사용할 IAM 역할을 생성합니다.

```
./deploy-role.sh workshop
```

[결과 확인]

AWS Systems Manager --> 애플리케이션 도구->파라미터 스토어에서 파라미터 생성 확인

Lambda->추가 리소스->계층에 Lambda Layer 생성 확인

(이 실습은 Administration 계정에서 진행한다)

AWS CodeBuild는 완전관리형 빌드 서비스로, 소스 코드를 컴파일하고, 테스트를 실행하며, 배포 가능한 아티팩트를 생성하는 자동화 도구이다. Jenkins나 GitHub Actions과 같은 CI/CD 톨과 유사한 역할을 하지만, AWS에서 관리해주는 서버리스 구조로 동작한다.

1. 주요 특징

- **서버리스(Serverless)**

빌드 서버를 직접 구축하거나 관리할 필요 없이 CodeBuild가 컨테이너 기반으로 빌드 환경을 실행하고 종료한다.

- **자동 확장성**

동시에 여러 빌드 실행이 가능하며, 대기 시간이 거의 없다.

- **비용 효율성**

사용한 시간(분 단위)만큼만 과금되며, EC2 인스턴스를 상시 구동할 필요가 없다.

- **보안 통합**

IAM, VPC, KMS, CloudTrail 등 AWS 보안 서비스와 쉽게 통합된다.

2. 기본 작동 흐름

1. 소스 저장소 연동

GitHub, Bitbucket, AWS CodeCommit, Amazon S3 등에서 소스를 가져온다.

2. 빌드 사양(buildspec.yml) 실행

빌드 명령은 buildspec.yml 파일에 정의하며, CodeBuild는 이 파일을 기준으로 동작한다.

3. 컨테이너 기반 빌드 실행

제공된 이미지 또는 커스텀 Docker 이미지로 컨테이너가 실행된다.

4. 출력 아티팩트 생성

컴파일된 결과물(예: JAR, ZIP, Docker 이미지 등)을 S3, ECR 등에 업로드한다.

5. 통합 사용

CodePipeline, CloudWatch, SNS, Lambda 등과 연동해 CI/CD 또는 모니터링을 구축할 수 있다.

3. buildspec.yml 구조 예시

version: 0.2

phases:

install:

runtime-versions:

java: corretto11

commands:

- echo Installing dependencies...
- ./gradlew clean

build:

commands:

- echo Building the project...
- ./gradlew build

artifacts:

files:

- build/libs/*.jar

discard-paths: yes

4. 주요 설정 항목

항목	설명
source	GitHub, CodeCommit 등 소스 위치 지정
environment	실행할 런타임 환경 (OS, 언어, Docker 이미지 등)
buildspec	빌드 명령 정의 파일
artifacts	빌드 결과물 출력 위치
logs	CloudWatch Logs 연동 설정

5. 활용 예시

- Java 프로젝트를 빌드하고 S3에 JAR 파일 업로드
 - Docker 이미지를 빌드하고 Amazon ECR에 푸시
 - AWS Lambda 함수 코드를 빌드하고 자동 배포
 - GitHub PR 마다 자동 빌드 테스트 수행
-

6. 자주 사용하는 통합 대상

- **AWS CodePipeline:** CI/CD 파이프라인 구축
- **Amazon S3:** 정적 웹사이트 배포 아티팩트 저장
- **Amazon ECR:** Docker 컨테이너 이미지 저장소
- **CloudWatch Logs:** 빌드 로그 저장
- **IAM 역할:** 소스 읽기, 아티팩트 저장, 로그 기록 권한 관리

AWS Systems Manager는 AWS에서 제공하는 **인프라 운영 관리 서비스**이다. EC2 인스턴스, 온프레미스 서버, 기타 AWS 리소스를 **중앙에서 통합 관리**할 수 있도록 도와준다. 대표적으로 패치 관리, 구성 관리, 자동화, 원격 셸 접속 등을 제공하며, DevOps나 운영팀에서 자주 사용된다.

1. 주요 기능 요약

기능	설명
Session Manager	EC2 인스턴스에 SSH 없이 웹 콘솔 또는 CLI로 안전하게 접속
Automation	반복 작업(예: AMI 생성, 인스턴스 재시작)을 문서화해서 자동 수행
Run Command	인스턴스 그룹에 명령어를 원격 실행
Patch Manager	OS 패치 자동화 및 정책 기반 패치 관리
Parameter Store	비밀번호, API 키 등의 설정값을 안전하게 저장 및 조회
Inventory	인스턴스의 설치된 소프트웨어, 설정 등 자산 정보 수집
State Manager	인스턴스 구성 상태를 일정하게 유지 (예: 에이전트 자동 설치 등)
OpsCenter	운영 이슈 티켓 모니터링 및 해결 조치 추적

2. 작동 방식

1. SSM 에이전트 설치

- EC2 또는 온프레미스 서버에 **SSM Agent**가 설치되어 있어야 한다.
- 대부분의 Amazon Linux / Ubuntu는 기본 설치되어 있다.

2. IAM 역할 부여

- SSM이 동작하려면 인스턴스에 적절한 IAM 역할 (AmazonSSMManagedInstanceCore 등)이 필요하다.

3. Systems Manager 콘솔 또는 CLI로 제어

- 웹 콘솔, AWS CLI, SDK에서 명령어 실행, 자동화 워크플로우 관리 등이 가

능하다.

3. 주요 사용 예

✔ Session Manager

- EC2에 SSH 포트를 열지 않고, IAM 인증만으로 브라우저 기반 셸 접속
- 보안그룹에서 포트 22 안 열어도 됨

✔ Run Command

- 수십, 수백 개 인스턴스에 동시에 명령어 실행 가능
- 예: "yum update -y" 또는 "systemctl restart nginx"

✔ Patch Manager

- 운영체제별로 패치 정책 설정 후, 정기적으로 자동 패치 적용

✔ Automation

- YAML/JSON 문서로 작업 정의
- 예:
 - EC2 백업 및 종료
 - 특정 태그가 달린 인스턴스만 재시작

✔ Parameter Store

- 민감한 설정값을 저장하고, Lambda, EC2, ECS 등에서 호출하여 사용
- 암호화 옵션 제공 (KMS 연동 가능)

4. 실습 예: EC2에 SSH 없이 접속 (Session Manager)

1. EC2에 다음 IAM 역할 부여:
 - AmazonSSMManagedInstanceCore
2. 보안 그룹에서 포트 22 **달아도 됨**
3. AWS 콘솔 > Systems Manager > Session Manager > "Start session" 클릭

5. 요금

- 대부분의 기능은 **무료** (Parameter Store 고급 매개변수, Automation 고급 문서 등은 유료)
- 로그 저장, KMS 사용, 고급 기능만 별도 과금

AWS Systems Manager의 **Parameter Store**는 애플리케이션 설정값(예: DB 접속 정보, API 키, 구성 플래그 등)을 **중앙에서 안전하게 저장하고 관리**할 수 있는 서비스이다. 다른 AWS 서비스(Lambda, EC2, ECS 등)와 연동하여 설정값을 안전하게 주입하거나 조회할 수 있다.

1. Parameter Store의 주요 특징

항목	설명
계층적 키 저장	"/app/dev/db_password" 같은 경로 형태로 구성 가능
보안 저장	평문 또는 KMS로 암호화 하여 저장 가능
버전 관리	파라미터 수정 시 버전이 올라가며, 이전 버전으로 롤백 가능
IAM 제어	IAM 정책으로 파라미터 접근 권한 제어 가능
이벤트 통합	파라미터 변경 시 CloudWatch Events로 감지 가능
저비용 또는 무료	표준 파라미터는 무료, 고급 파라미터는 유료

2. 파라미터 유형 (Type)

유형	설명
String	일반 텍스트 (예: URL, 경로 등)
StringList	쉼표로 구분된 문자열 목록 (예: IP 리스트)
SecureString	암호화된 문자열 (기본 KMS 또는 사용자 지정 KMS 키로 암호화 가능)

3. 사용 예시

✅ 콘솔에서 생성

- Systems Manager → Parameter Store → "Create Parameter"
- 예:
 - Name: /prod/db/username
 - Type: SecureString
 - Value: admin_user
 - KMS Key: (기본 AWS 키 사용 가능)

✅ AWS CLI에서 생성

```
aws ssm put-parameter ₩  
  --name "/prod/db/password" ₩  
  --value "SuperSecret123!" ₩  
  --type "SecureString" ₩  
  --overwrite
```

✅ AWS CLI에서 조회

```
aws ssm get-parameter ₩  
  --name "/prod/db/password" ₩  
  --with-decryption
```


4. Parameter Store를 사용하는 서비스 예

서비스	활용 방법
EC2	사용자 데이터나 애플리케이션 코드에서 설정값을 가져옴
Lambda	파라미터를 환경 변수처럼 로딩하여 민감 정보 관리
ECS / Fargate	Task 정의에서 파라미터를 환경 변수로 주입 가능
CloudFormation	템플릿 내 설정값을 Parameter Store에서 불러오기 가능
CodePipeline / CodeBuild	빌드 중 파라미터를 참조하여 인증 정보 주입

5. Parameter Store vs Secrets Manager

항목	Parameter Store	Secrets Manager
기본 요금	표준 파라미터는 무료	유료 (\$0.40/비밀/월)
자동 교체	직접 구현해야 함	자동 비밀번호 교체 기능 내장
접근 방식	SSM API 또는 SDK	Secrets Manager 전용 API
대상	일반 구성값, 비밀 값 둘 다	주로 비밀값(DB 암호, API 키)

6. 보안 팁

- 민감한 값은 반드시 SecureString + KMS 암호화 사용
- IAM 정책으로 접근 권한 최소화
- CloudTrail로 Parameter 접근 로그 추적 가능
- 파라미터 이름은 계층적 네이밍 규칙(/env/app/key) 사용하면 관리 편함

AWS Lambda Layer는 여러 Lambda 함수에서 **공통으로 사용하는 코드나 라이브러리, 설정 파일** 등을 **재사용 가능한 형태로 분리**해 놓은 구조이다. Python, Node.js 등에서 외부 라이브러리를 각 함수마다 포함시키지 않고, 공통 Layer로 올려두면 효율적으로 관리할 수 있다.

1. Lambda Layer의 개념

Lambda 함수는 기본적으로 **함수 코드 + 의존 파일**을 패키징해서 배포한다.

하지만 여러 함수에서 동일한 라이브러리(예: requests, pandas)나 모듈을 쓸 경우, **코드 중복**과 **배포 관리의 복잡성**이 생긴다.

→ 이때 Lambda Layer를 사용하면 공통 코드를 따로 분리하고, 여러 함수에서 공유해서 사용할 수 있다.

2. 주요 특징

항목	설명
재사용성	여러 Lambda 함수에서 같은 Layer를 참조 가능
버전 관리	Layer는 버전 단위로 관리되며, 롤백도 가능
권한 제어	계정 내부 공유 또는 공개 공유 가능
최대 5개	Lambda 함수당 최대 5개의 Layer를 연결할 수 있음
크기 제한	Layer 압축 기준 50MB (함수 패키지 포함 최대 250MB)

3. Layer 구성 방식

Layer는 특정 언어별 런타임 경로에 맞춰 구조화되어야 한다.

✅ 예: Python용 Layer 디렉터리 구조

```
python/  
└── lib/  
    └── python3.9/  
        └── site-packages/
```

└─── <라이브러리 파일들>

압축해서 .zip 파일로 만든 후 업로드한다.

4. Layer 만들기 예시 (Python)

✅ 1. 의존 라이브러리 설치

```
mkdir python
```

```
pip install requests -t python/
```

```
zip -r layer.zip python/
```

✅ 2. AWS CLI로 Layer 생성

```
aws lambda publish-layer-version ₩
  --layer-name my-common-libs ₩
  --description "Common libraries for Lambda" ₩
  --zip-file fileb://layer.zip ₩
  --compatible-runtimes python3.9
```

✅ 3. Lambda 함수에 Layer 추가

Lambda 함수 설정에서 **"Layers → Add a layer"**

또는 CLI로 추가:

```
aws lambda update-function-configuration ₩
  --function-name my-function ₩
  --layers arn:aws:lambda:<region>:<account>:layer:my-common-libs:<version>
```

5. 사용 예시

- boto3, requests, pandas, numpy 등 공통 라이브러리 모듈 공유
 - 공통 유틸리티 함수 모음 (예: 로깅, 공통 포맷터 등)
 - 머신러닝 모델 바이너리 및 추론 코드
-

6. 주의 사항

항목	주의할 점
라이브러리 호환성	Layer에서 설치한 라이브러리의 Python 버전이 함수 런타임과 맞아 야 함
버전 관리	Layer 수정 시마다 새 버전 생성 필요
경량화 필요	함수 + Layer 합산 크기가 250MB 넘으면 실행 불가
네임스페이스 충돌	여러 Layer에서 동일 모듈 존재 시 충돌 가능

7. 예시: Layer에서 라이브러리 사용 (Python)

Layer에 requests가 들어 있다면, Lambda 함수에서는 그냥 다음처럼 쓰면 된다:

```
import requests
```

```
def lambda_handler(event, context):  
    r = requests.get("https://api.example.com")  
    return r.status_code
```

스토리지 계층 배포

배포 단계

```
mkdir sdlf-workshop
```

```
cd sdlf-workshop/
```

실습 주의 사항

만일 다른 실습에서 사용한 스택을 삭제하지 않으면 아래 명령 수행 시 오류가 발생한다

```
aws cloudformation deploy --template-file ./foundations-datalake-dev.yaml --stack-name  
sdlf-foundations-datalake-dev --capabilities "CAPABILITY_NAMED_IAM"  
"CAPABILITY_AUTO_EXPAND"
```

<원인>

데이터 레이크 관리자(DataLakeAdmins)가 lf-data-admin으로만 되어 있어서 IAM 사용자 lf-data-admin만 LakeFormation 설정을 변경할 수 있는 권한을 가지고 있다

Admin 계정에서도 권한 부여 불가 (lf-data-admin으로 로그인해야 설정 가능)

coud shell 권한 확인 알아보는 명령

```
sdlf-workshop $ aws lakeformation get-data-lake-settings  
{  
  "DataLakeSettings": {  
    "DataLakeAdmins": [  
      {  
        "DataLakePrincipalIdentifier": "arn:aws:iam::891377038690:user/lf-  
data-admin"  
      }  
    ],  
    ...  
  }  
}
```

[Lake Formation 권한 새로 추가]

lf-data-admin만 Lake Formation 권한을 가지고 있기 때문에 권한 추가를 한다

계속 Administrator 계정에서 진행한다

AWS Lake Formation->Administration->Administrative roles and tasks -> 예 가서

Manage administrators를 클릭하고 Add administrators에서

IAM users and roles 에 sdlf-cicd-codebuild-891377038690-workshop 을 선택해준다

Add administrators

Access type

☒ Data lake administrator

IAM users and roles
Add or remove IAM users and roles from the data lake administrators list.

Choose IAM principals to add

sdlf-cicd-codebuild-891377038690-workshop X
Role

Choose up to a maximum of 30 data lake administrators.

Active Directory and Amazon QuickSight users and groups, and federated users
Enter an Active Directory ARN (EMR beta only), Amazon QuickSight ARN, or federated user ARN. Press Enter to add.

Ex: `arn:aws:iam::<AccountId>:saml-provider/<SamlProviderName>:user/<UserName>`

Data lake administrators (1/1)				
Administrators can view all metadata in the Data Catalog. They can also grant and revoke permissions on data resources to principals, including themselves.				
<input type="text" value="Find administrators"/>				
<input type="checkbox"/>	Name	Status	Type	Access type
<input type="checkbox"/>	sdlf-cicd-codebuild-891377038690-workshop	Valid	IAM role	Full

편집기로 nano 편집기를 사용하면 편하다

nano foundations-datalake-dev.yaml

스크립트 복사하여 붙여 넣고 Ctrl-O -> Enter -> Ctrl-X 를 누르면 된다

원본이 오류 나므로 아래 오류 수정본을 사용한다

<원본>

```

AWSTemplateFormatVersion: "2010-09-09"
Description: SDLF Foundations in datalake domain, dev environment

Parameters:
  pPipelineReference:
    Type: String
    Default: none

Resources:
  rAnycompany:
    Type: AWS::CloudFormation::Stack
    Properties:

```

```
TemplateURL: "{{resolve:ssm:/sdlf/foundations/main}}"
Parameters:
  pPipelineReference: !Ref pPipelineReference
  pChildAccountId: !Ref AWS::AccountId
  pOrg: anycompany
  pDomain: datalake
  pDeploymentInstance: dev
  pCicdRole: Admin
```

<오류 수정본>

```
AWSTemplateFormatVersion: "2010-09-09"
Description: SDLF Foundations in datalake domain, dev environment

Parameters:
  pPipelineReference:
    Type: String
    Default: none

  pCicdRole:
    Type: String
    Default: sdlf-cicd-codebuild-891377038690-workshop
    Description: IAM Role to be used as the Lake Formation admin

Resources:
  rAnycompany:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: "{{resolve:ssm:/sdlf/foundations/main}}"
      Parameters:
        pPipelineReference: !Ref pPipelineReference
        pChildAccountId: !Ref AWS::AccountId
        pOrg: anycompany
        pDomain: datalake
        pDeploymentInstance: dev
        pCicdRole: !Ref pCicdRole
```

cloudformation 배포 명령

<오류 수정본>

```
aws cloudformation deploy \
  --region us-east-1 \
  --template-file foundations-datalake-dev.yaml \
  --stack-name sdlf-foundations-datalake-dev \
```

```
--capabilities CAPABILITY_NAMED_IAM CAPABILITY_AUTO_EXPAND \
--parameter-overrides \
  pPipelineReference=none \
  pCicdRole=sdlf-cicd-codebuild-891377038690-workshop
```

stack 생성 결과 확인

중첩
sdlf-foundations-datalake-dev-rAnycompany-1MELGHWLQHF8Y
2025-04-12 20:54:25 UTC+0900
CREATE_COMPLETE
sdlf-foundations-datalake-dev
2025-04-12 20:54:17 UTC+0900
CREATE_COMPLETE
sdlf-cfn-template-stageglue
2025-04-12 19:53:14 UTC+0900
CREATE_COMPLETE
sdlf-cfn-template-stagelambda
2025-04-12 19:52:29 UTC+0900
CREATE_COMPLETE
sdlf-cfn-template-team
2025-04-12 19:51:44 UTC+0900
CREATE_COMPLETE

이벤트 (6)			
이벤트 검색			
타임스탬프	논리적 ID	상태	세
2025-04-12 20:56:19 UTC+0900	sdlf-foundations-datalake-dev	CREATE_COMPLETE	-
2025-04-12 20:56:18 UTC+0900	rAnycompany	CREATE_COMPLETE	-
2025-04-12 20:54:25 UTC+0900	rAnycompany	CREATE_IN_PROGRESS	-
2025-04-12 20:54:24 UTC+0900	rAnycompany	CREATE_IN_PROGRESS	-
2025-04-12 20:54:22 UTC+0900	sdlf-foundations-datalake-dev	CREATE_IN_PROGRESS	-
2025-04-12 20:54:17 UTC+0900	sdlf-foundations-datalake-dev	REVIEW_IN_PROGRESS	-

S3 버킷 생성 결과 확인

범용 버킷 (15) 정보 모든 AWS 리전				
버킷은 S3에 저장되는 데이터의 컨테이너입니다.				
이름으로 버킷 찾기				
이름	AWS 리전	IAM Access Analyzer	생성 날짜	
anycompany-datalake-us-east-1-891377038690-analytics-dev	미국 동부(버지니아 북부) us-east-1	us-east-1에 대한 분석기 보기	2025. 4. 12. pm 8:55:25 PM KST	
anycompany-datalake-us-east-1-891377038690-artifacts-dev	미국 동부(버지니아 북부) us-east-1	us-east-1에 대한 분석기 보기	2025. 4. 12. pm 8:55:25 PM KST	
anycompany-datalake-us-east-1-891377038690-athena-dev	미국 동부(버지니아 북부) us-east-1	us-east-1에 대한 분석기 보기	2025. 4. 12. pm 8:55:26 PM KST	
anycompany-datalake-us-east-1-891377038690-raw-dev	미국 동부(버지니아 북부) us-east-1	us-east-1에 대한 분석기 보기	2025. 4. 12. pm 8:55:25 PM KST	
anycompany-datalake-us-east-1-891377038690-s3logs-dev	미국 동부(버지니아 북부) us-east-1	us-east-1에 대한 분석기 보기	2025. 4. 12. pm 8:55:10 PM KST	
anycompany-datalake-us-east-1-891377038690-stage-dev	미국 동부(버지니아 북부) us-east-1	us-east-1에 대한 분석기 보기	2025. 4. 12. pm 8:55:25 PM KST	

AWS에서 말하는 **Artifacts(아티팩트)**는 보통 빌드나 배포 과정에서 생성된 결과물을 의미한다. 특히 **AWS CodeBuild, CodePipeline, Lambda, S3, Amplify** 같은 CI/CD 또는 배포 관련 서비스에서 자주 등장한다.

1. Artifacts의 개념

아티팩트는 소스 코드를 컴파일하거나 처리한 후 산출된 출력 파일이다.

예를 들어:

입력	빌드 후 아티팩트
Java 소스코드	.jar, .war 파일
Node.js 앱	dist/ 폴더, 번들된 .js 파일
Python 패키지	.zip, .whl
웹 정적 파일	HTML, CSS, JS 번들
Docker 앱	Docker 이미지 (ECR로 업로드됨)

2. Artifacts가 사용되는 위치

서비스	아티팩트 역할
CodeBuild	빌드 후 결과물을 S3로 저장하거나, CodePipeline에 전달
CodePipeline	빌드 → 테스트 → 배포 단계 간 아티팩트를 전달
Lambda	.zip 또는 S3에서 불러온 패키지를 아티팩트로 간주
Amplify	프론트엔드 빌드 후 HTML/CSS/JS 파일을 아티팩트로 배포
S3	아티팩트를 저장하는 저장소로 자주 사용됨

3. CodeBuild에서의 예시

✅ **buildspec.yml 예시:**

```
version: 0.2
```

```
phases:
```

```
  build:
```

```
    commands:
```

```
- echo "Build complete"
```

artifacts:

files:

```
- dist/**
```

base-directory: dist

→ 위 설정은 dist/ 폴더에 있는 모든 파일을 빌드 아티팩트로 지정하고,

→ CodePipeline의 다음 단계나 S3로 넘긴다.

4. S3로 아티팩트 저장 예시 (CodeBuild)

artifacts:

files:

```
- '**/*'
```

discard-paths: yes

name: my-app-output

이렇게 하면 모든 파일을 루트에 저장하고, 이름은 my-app-output.zip 같은 식으로 된다.

5. 아티팩트 vs 소스코드

구분	설명
소스코드	개발자가 작성한 원본 코드 (예: .java, .ts, .py)
아티팩트	소스코드를 처리한 결과물 (예: jar, .zip, bundle.js)
역할	아티팩트를 배포하거나 테스트에 사용함

6. 다른 예: Lambda에서의 아티팩트

Lambda 함수 배포할 때 lambda_function.zip 파일을 만들어서 올린다면,

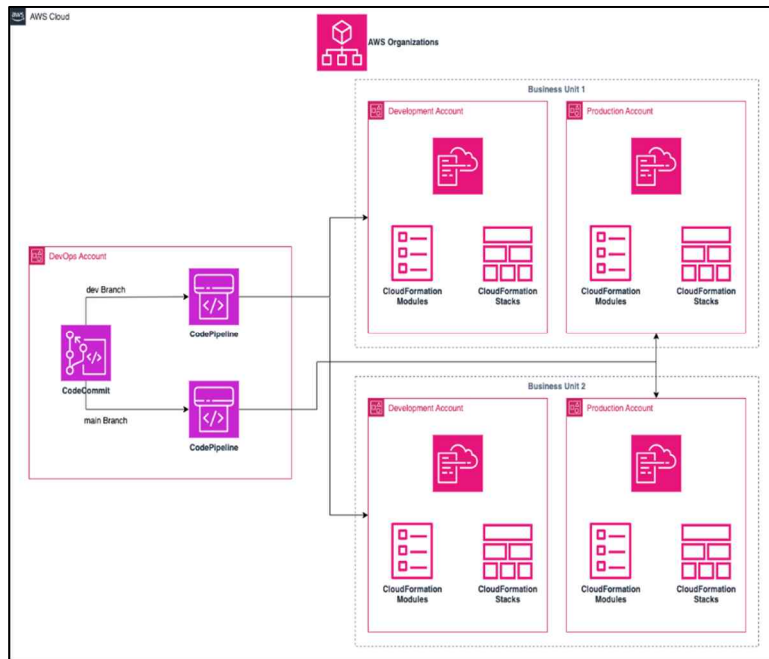
이 .zip 파일이 바로 **배포 아티팩트**이다.

CodePipeline이나 CodeBuild로 만들고 S3에 업로드해 사용 가능하다.

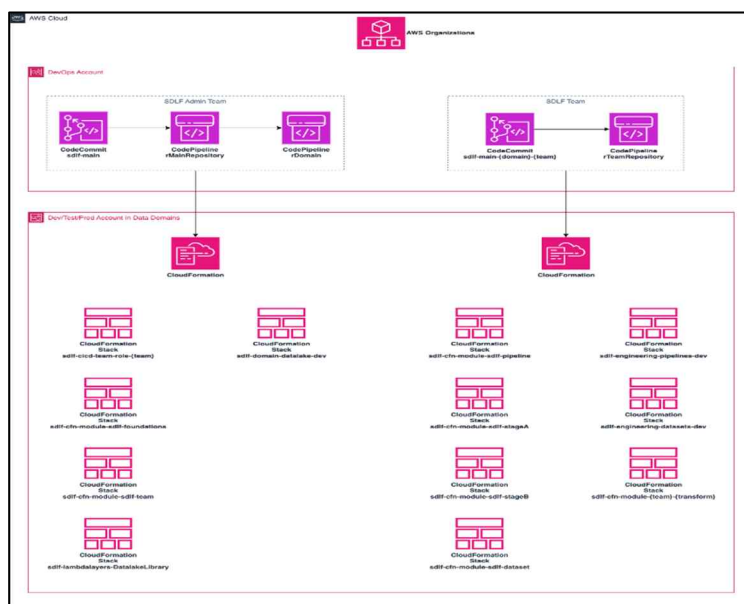
SDLF 배포 실습은 여기 까지만 진행(내부 템플릿 스크립트 버전 문제로 오류 발생!!)

Hitting production

중앙 집중식 데이터 레이크 및 분산형 데이터 도메인



CICD 아키텍처



필수 구성 요소

워크숍의 이 부분을 완료하려면 다음이 필요합니다.

- **DevOps 계정:** 사용되지 않은 새 AWS 계정
- **개발/테스트/Prod 계정의 첫 번째 집합:** 사용되지 않는 새 자식 AWS 계정(Dev 계정) 또는 최대 3개의 자식 계정(Dev, Test 및 Prod 계정)
- **두 번째 개발/테스트/Prod 계정 집합:** 사용되지 않는 새 하위 AWS 계정(Dev 계정) 또는 최대 3개의 하위 계정(Dev, Test 및 Prod 계정)

각 집합은 데이터 도메인이 됩니다. 데이터 도메인에는 최대 3개의 환경(개발, 테스트 및 프로덕션)이 있을 수 있습니다.

(실습 불가 : 실습 계정 부족함)