클라우드 데이터, 제대로 활용할 수 있는 3가지

Data Analytics Team,
Data Engineer 박 민지

Focus on Cloud
Microsoft Azure Consulting Expert Group

Cloocus

Gold Microsoft Partner
Microsoft

Azure Expert MSP

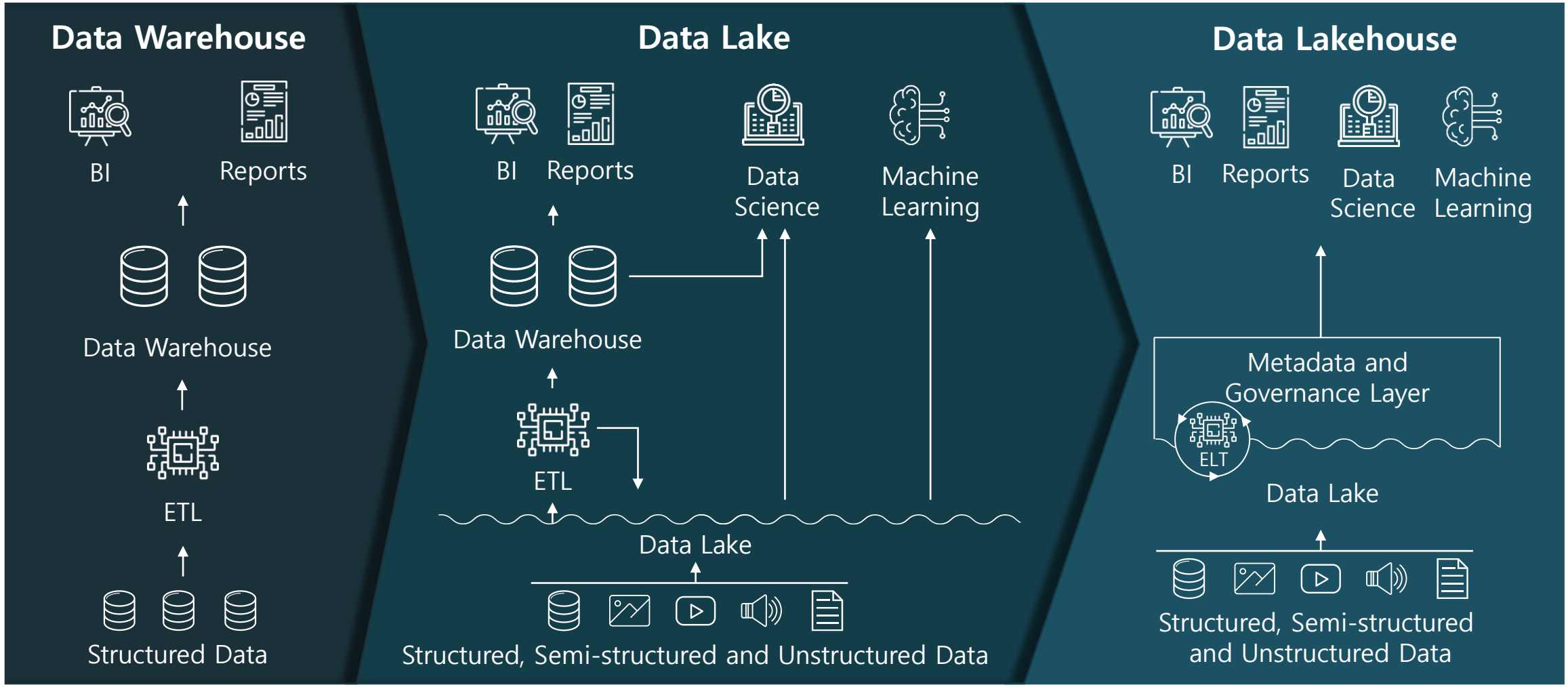
빅데이터 아키텍처 패러다임
- Introduction

빅데이터 아키텍처 패러다임 소개

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

빅데이터 아키텍처 워크로드 및 패러다임 소개

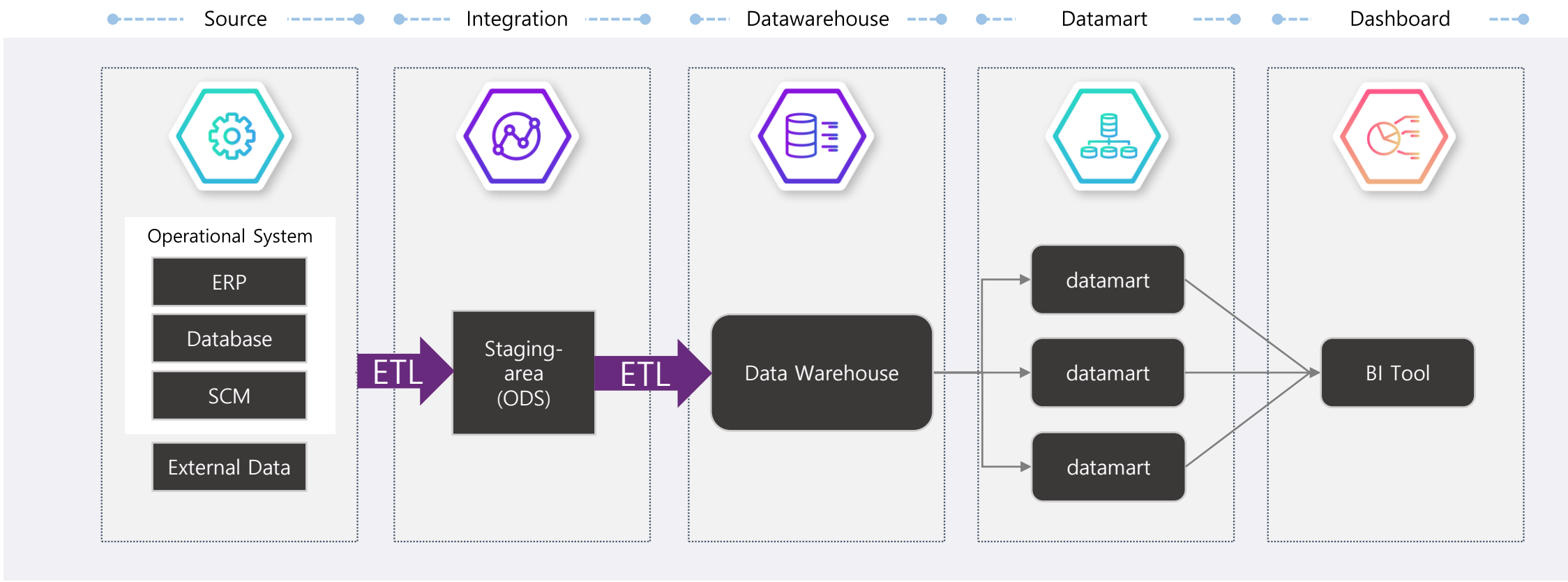


Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임
- Data Warehouse

Modern Data Warehouse

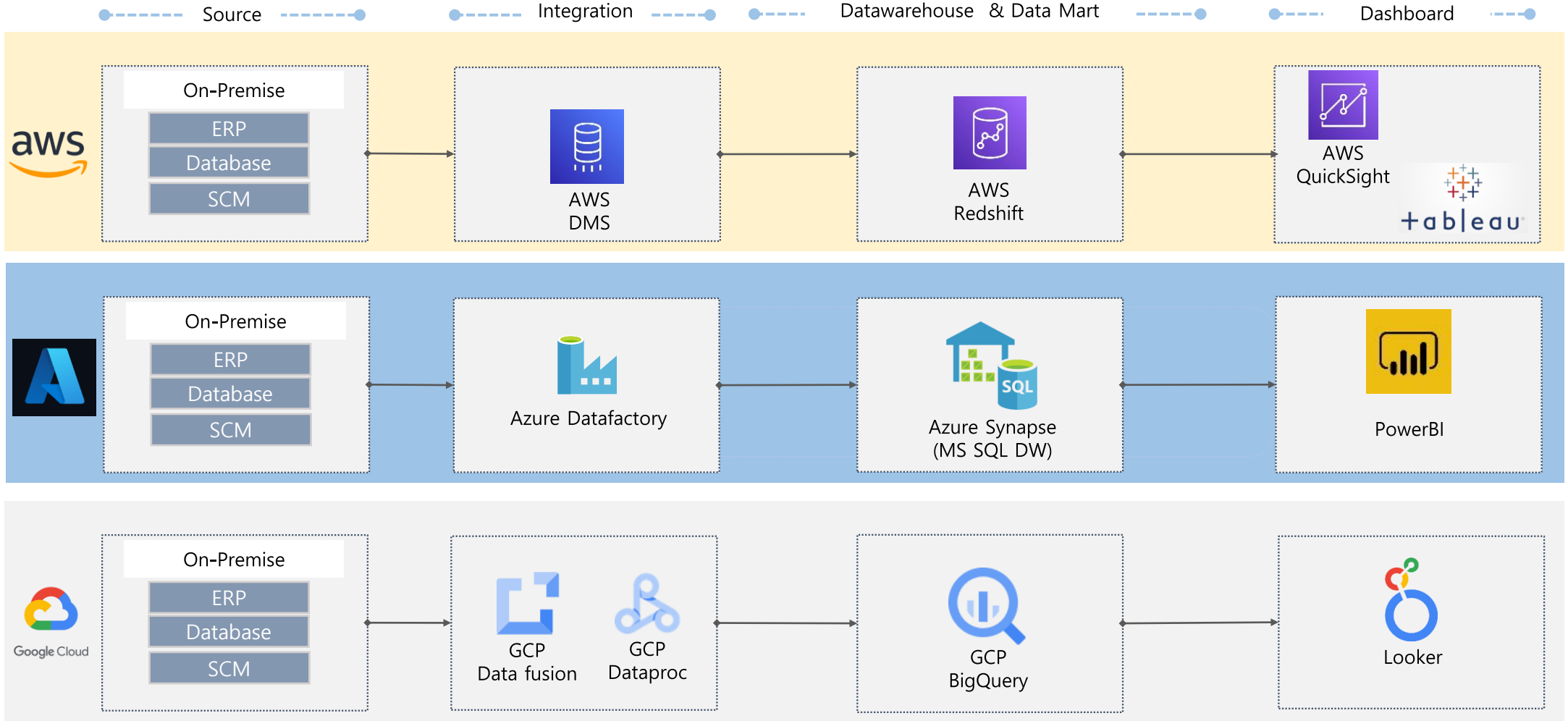
1. Modern Data Warehouse



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Data Warehouse on Cloud

2. Data Warehouse on AWS, Azure, GCP

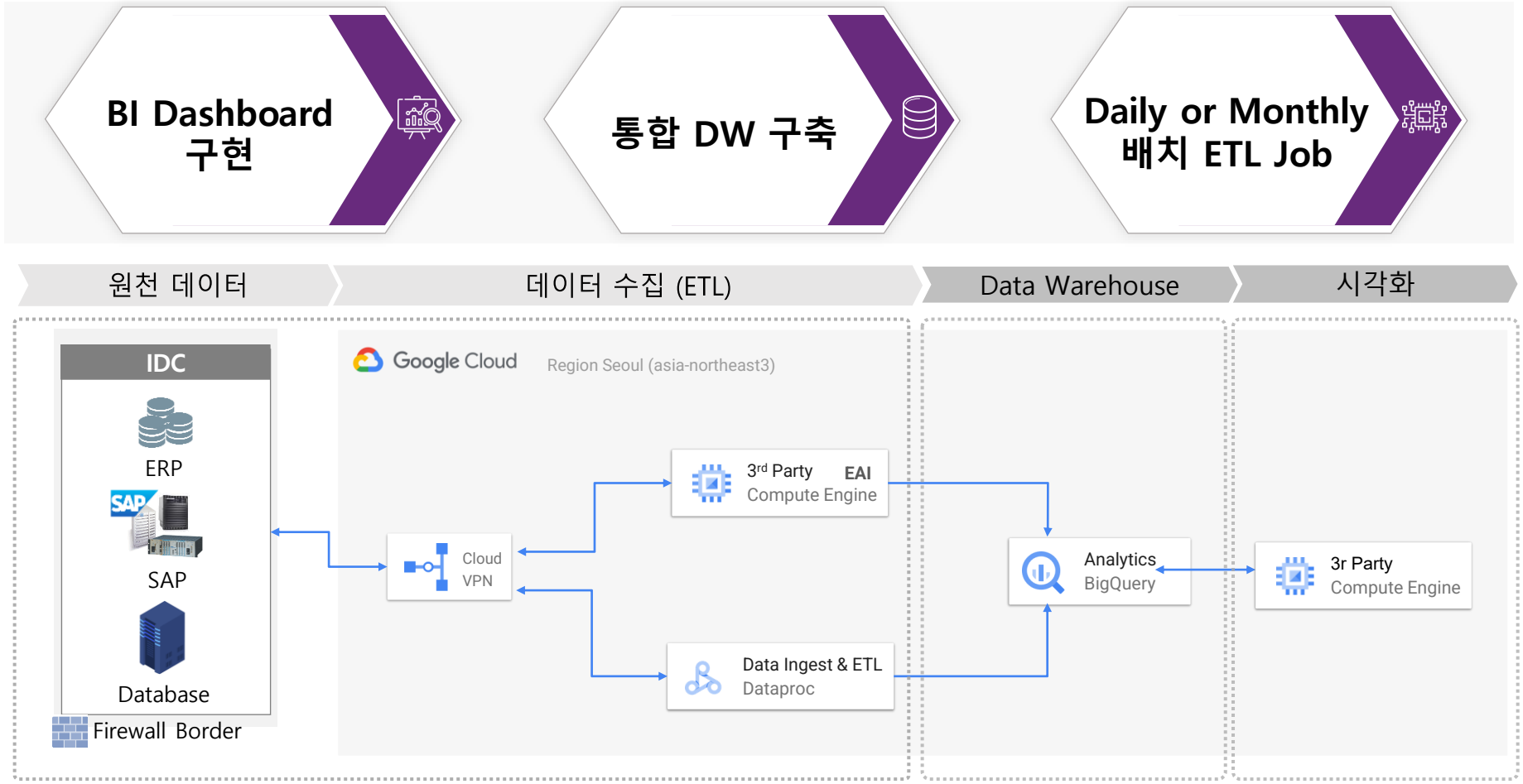


Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Data Warehouse on Cloud

3. 제조 Industry / K사 Refence – Data Warehouse

<Data Warehouse 구축 고려사항>



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Data Warehouse To Data Lake Architecture

4. Limits of Data Warehouse

1) Data Warehouse 구성 목표

: 정형 데이터를 기반으로 BI 및 Report 구현을 위함

2) Data Warehouse 제한

- 비정형 데이터 분석 요건의 경우
- AI 기반의 분석 요건의 경우
- 실시간 데이터 분석 요건의 경우

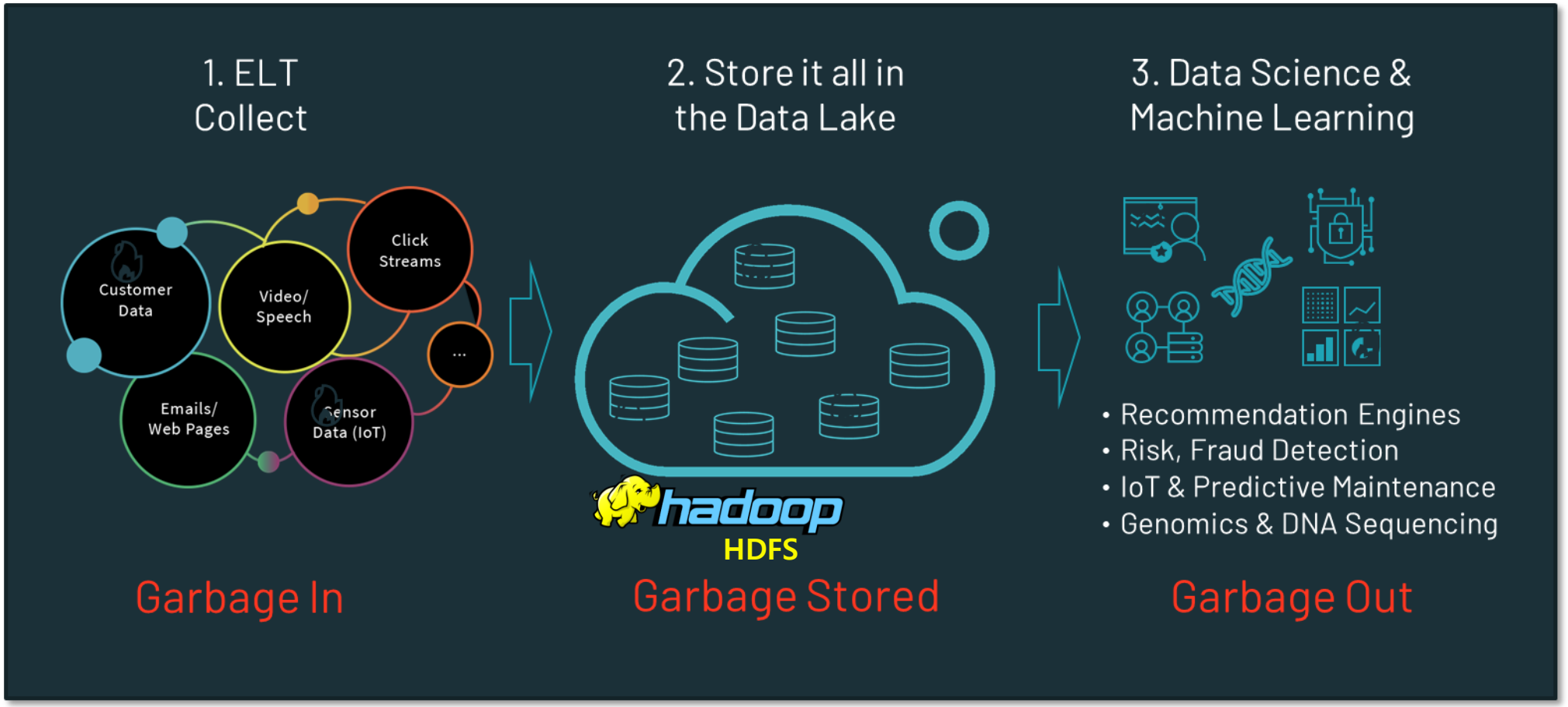
Data Lake 아키텍처 고려 必

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임
- Data Lake

빅데이터 아키텍처 패러다임

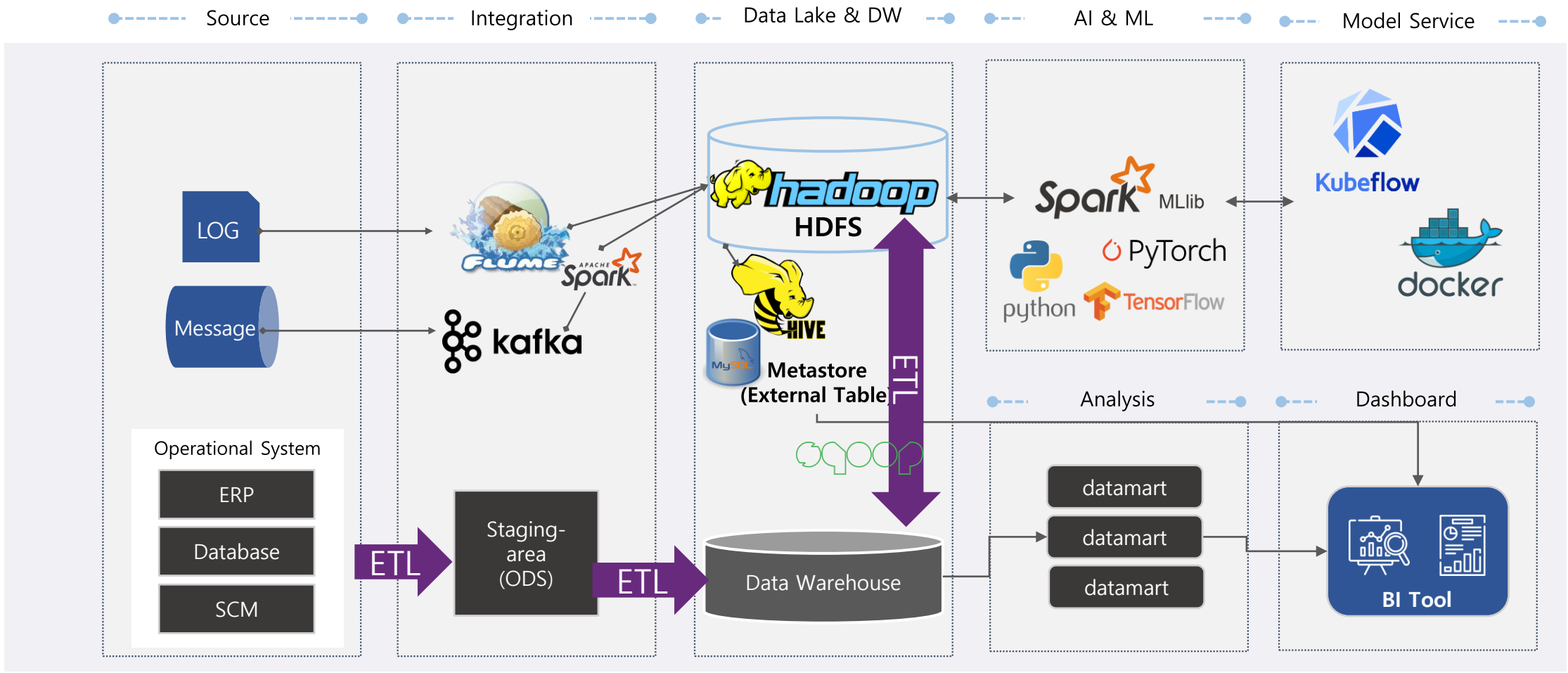
1. Concept of Data Lake Architecture



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

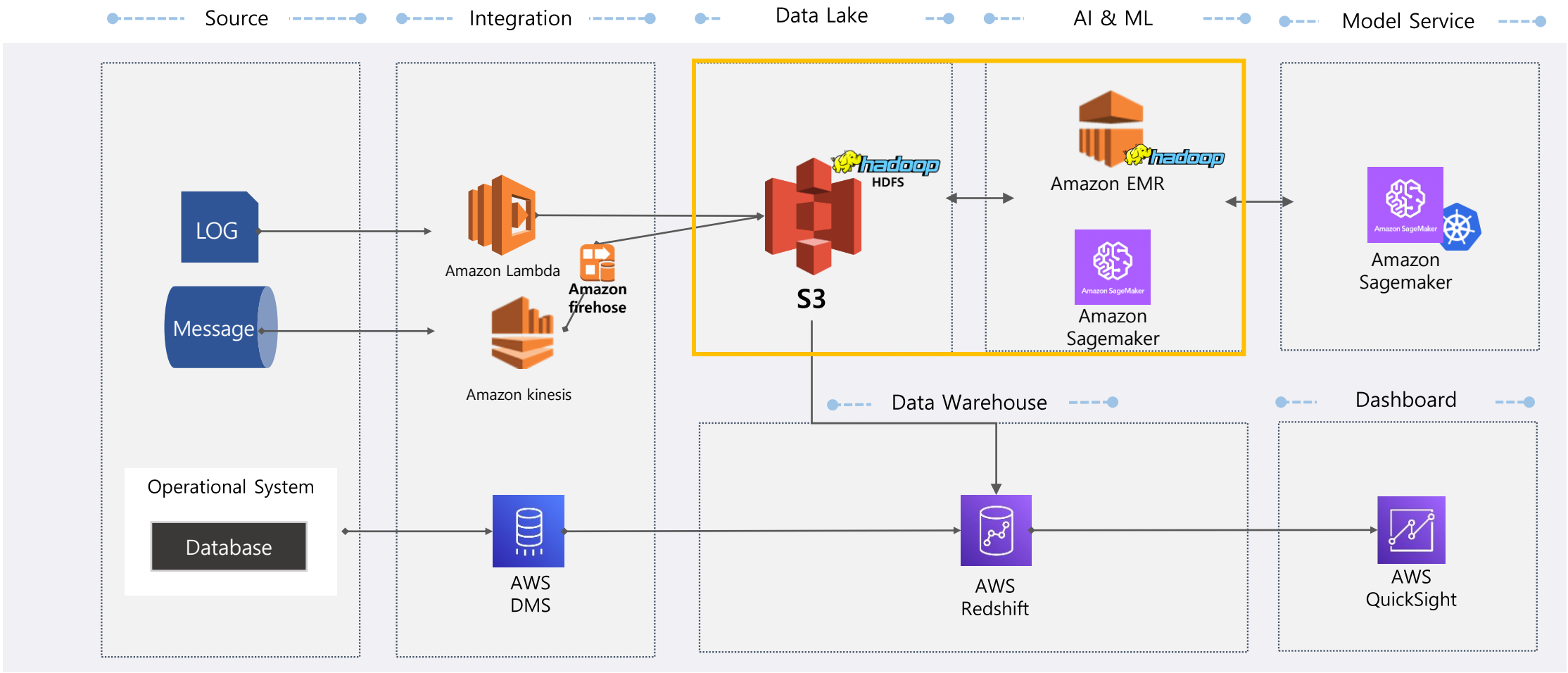
2. General Data Lake Architecture On-Premise – Hadoop Ecosystem + Data Warehouse



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

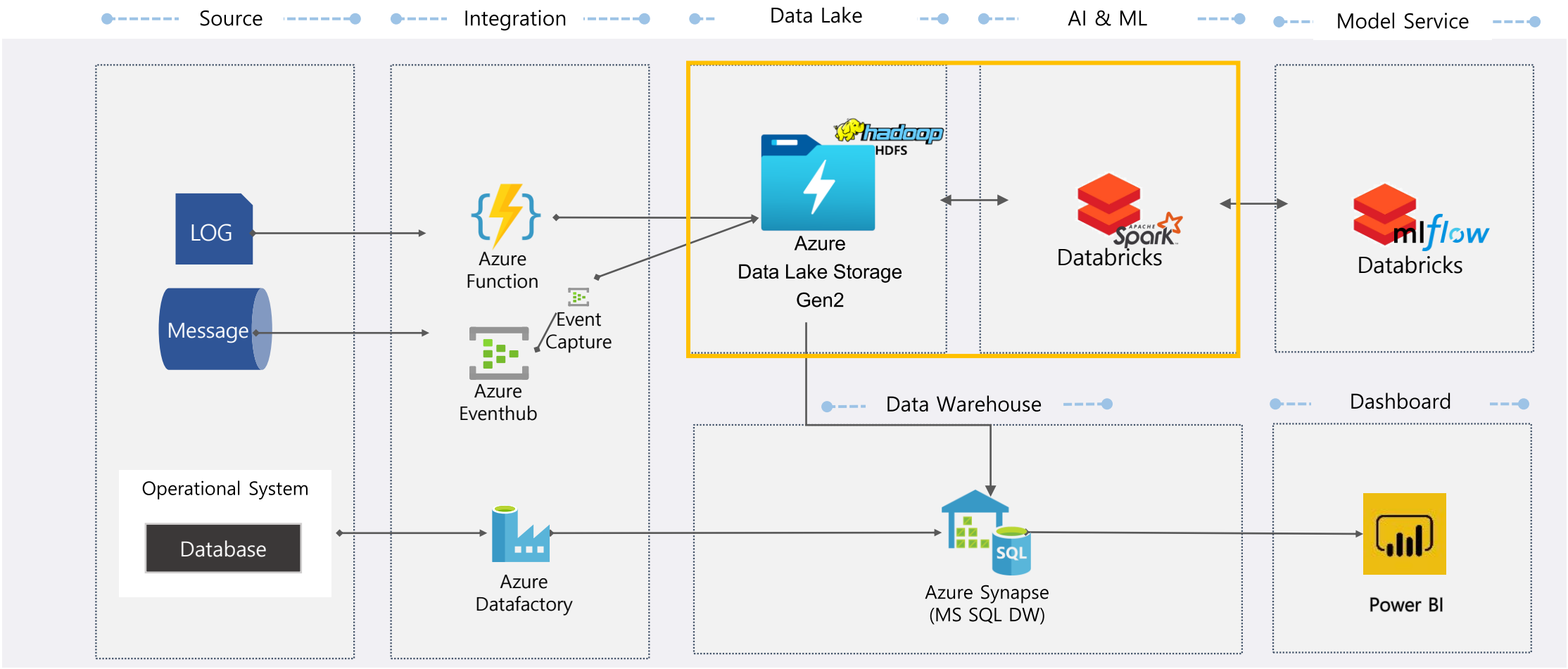
2. Data Lake Architecture on AWS



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

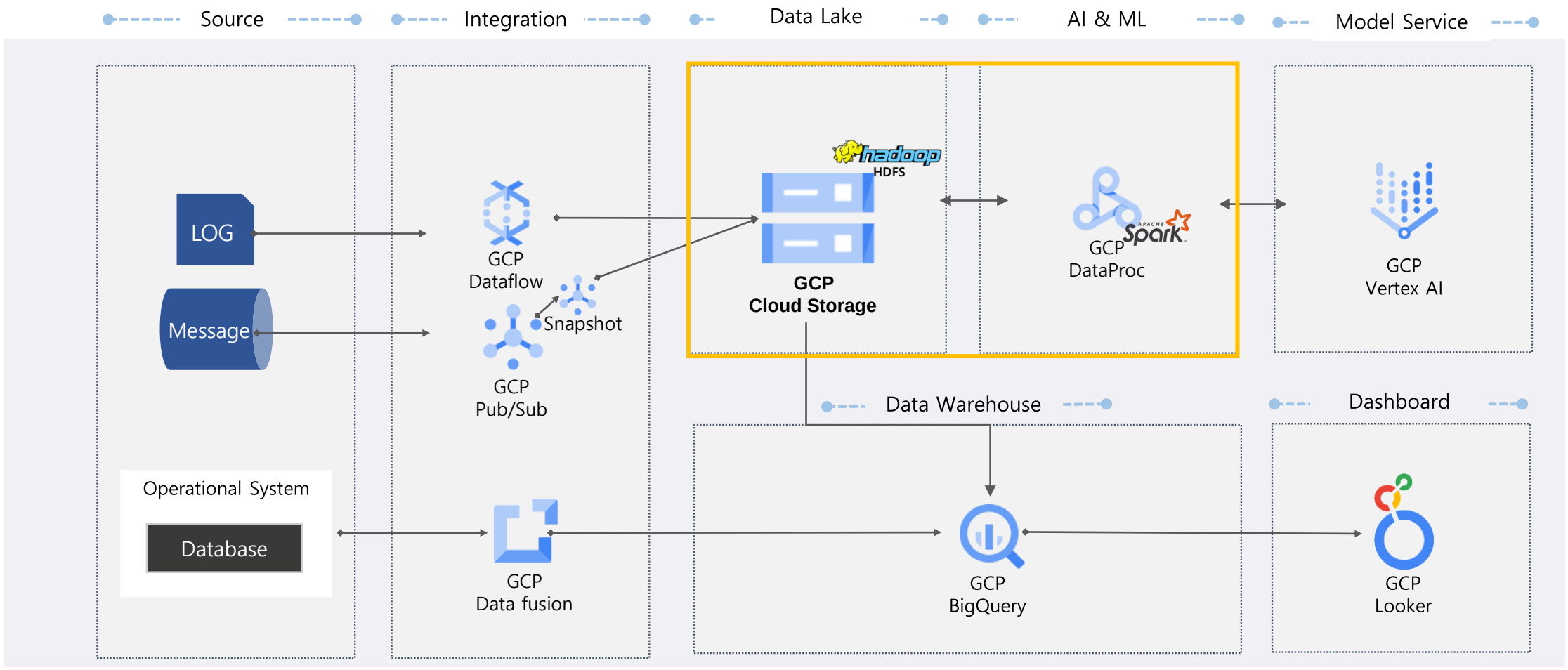
2. Data Lake Architecture on Azure



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

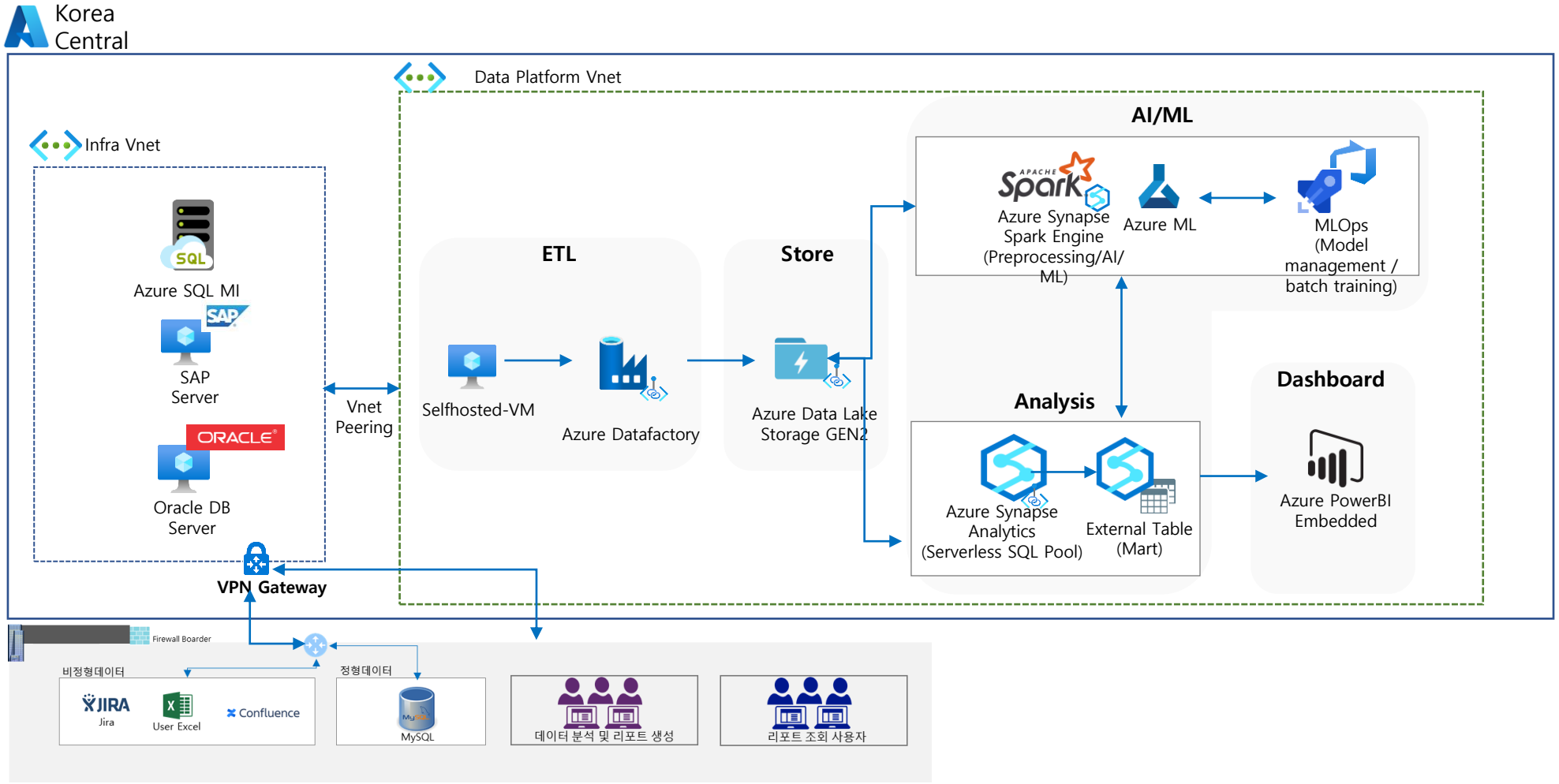
2. Data Lake Architecture on GCP



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

4. IT Industry / C사 Refence – Data Lake



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

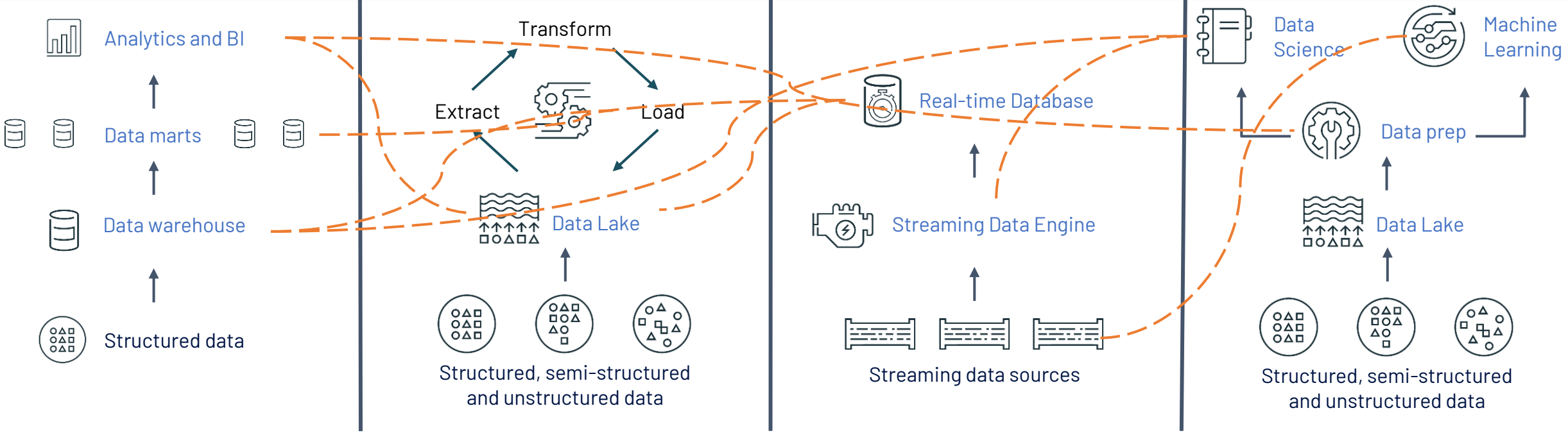
Data Lake Architecture
Challenge

Data Lake Architecture Challenge

연결되지 않은 시스템 및 독점적 데이터 포맷은 통합 난이도를 높임

Amazon Redshift	Teradata	Hadoop	Apache Airflow	Apache Kafka	Apache Spark	Jupyter	Amazon SageMaker
Azure Synapse	Google BigQuery	Amazon EMR	Apache Spark	Apache Flink	Amazon Kinesis	Azure ML Studio	MatLAB
Snowflake	IBM Db2	Google Dataproc	Cloudera	Azure Stream Analytics	Google Dataflow	Domino Data Labs	SAS
SAP	Oracle Autonomous Data Warehouse			Tibco Spotfire	Confluent	TensorFlow	PyTorch

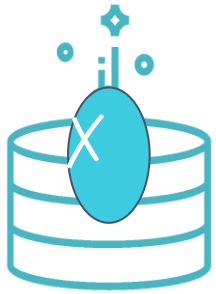
분리된 환경은 데이터 아키텍쳐의 복잡성을 증가시킴



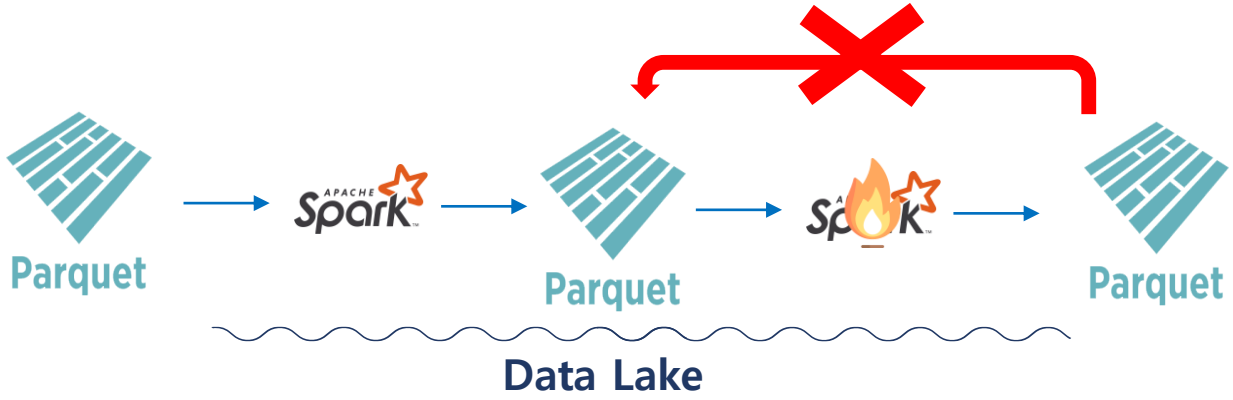
Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Data Lake Architecture Challenge

1. NO ACID



No atomicity

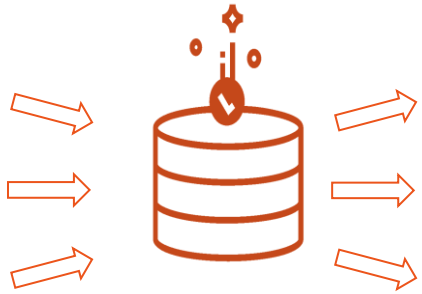


작업 과정 **중간**에 실패가 날 경우
손상된 데이터로 남아 복구하는데
많은 자원이 필요

Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

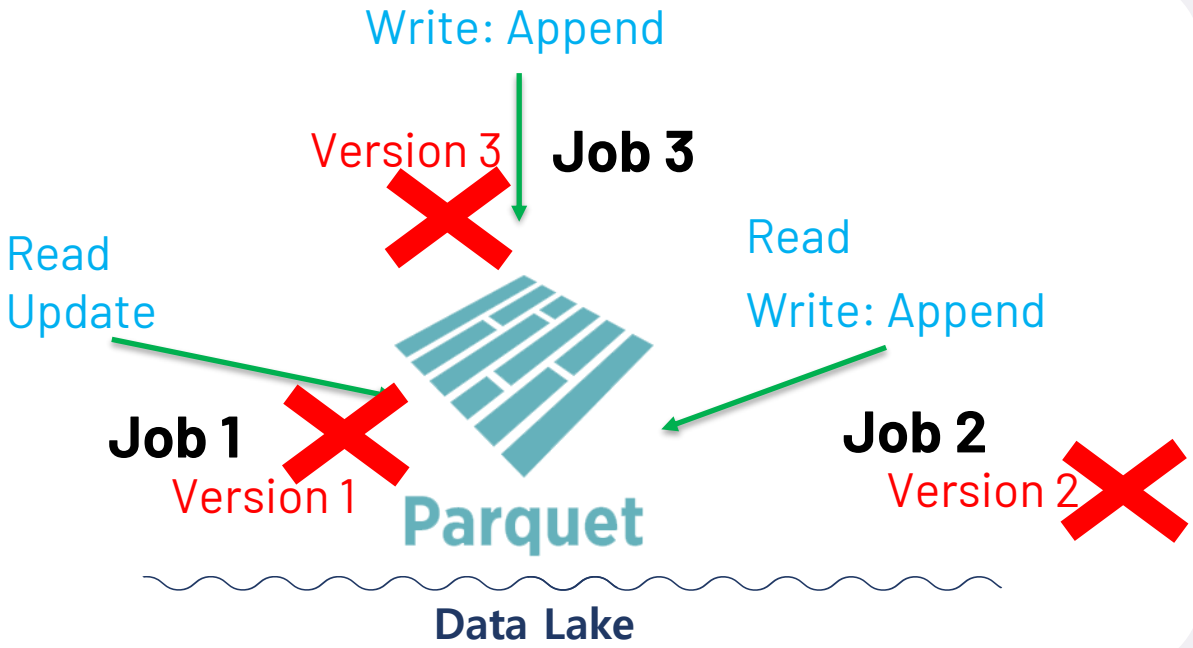
Data Lake Architecture Challenge

1. NO ACID



No consistency / isolation

Merge 및 Append,
배치 및 스트리밍을 혼합하는 것에
일관성과 고립성을 **보장하지 않음**



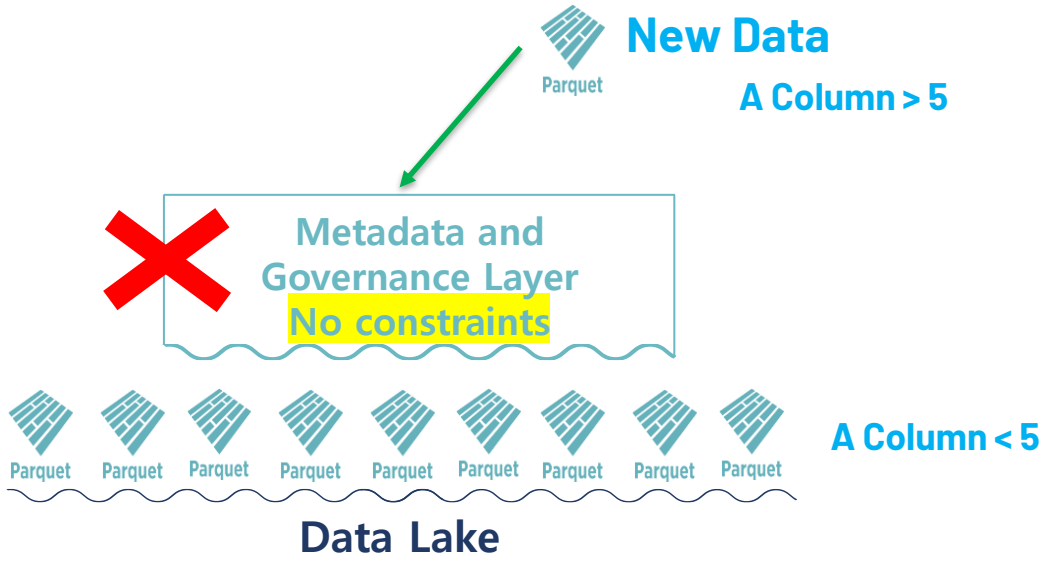
Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Data Lake Architecture Challenge

1. NO ACID

No quality enforcement

일관된 **데이터 품질 확인**의 어려움

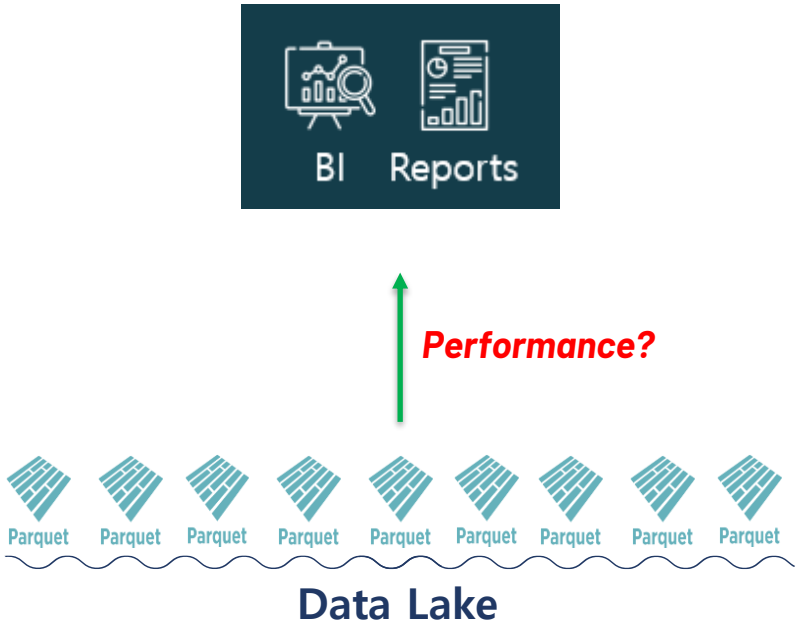


Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Data Lake Architecture Challenge

2. Data Warehouse Engine is Expensive

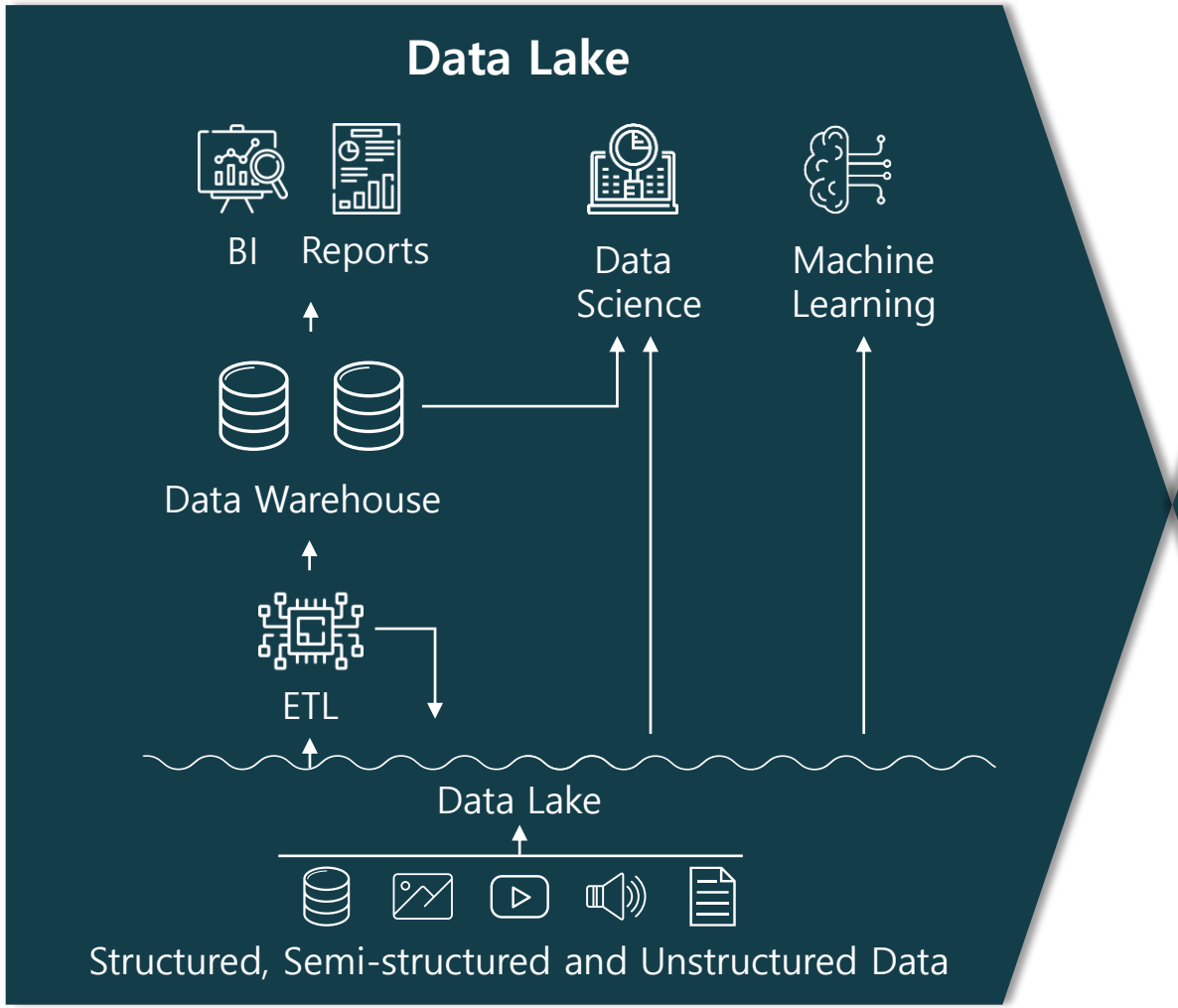
Needs of Data Warehouse



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

5. Limits of Data Lake Architecture



1) Data Lake 구성 장점

- 정형, 비정형, 반정형 데이터 로드
- AI&ML 분석

2) Data Lake 제한

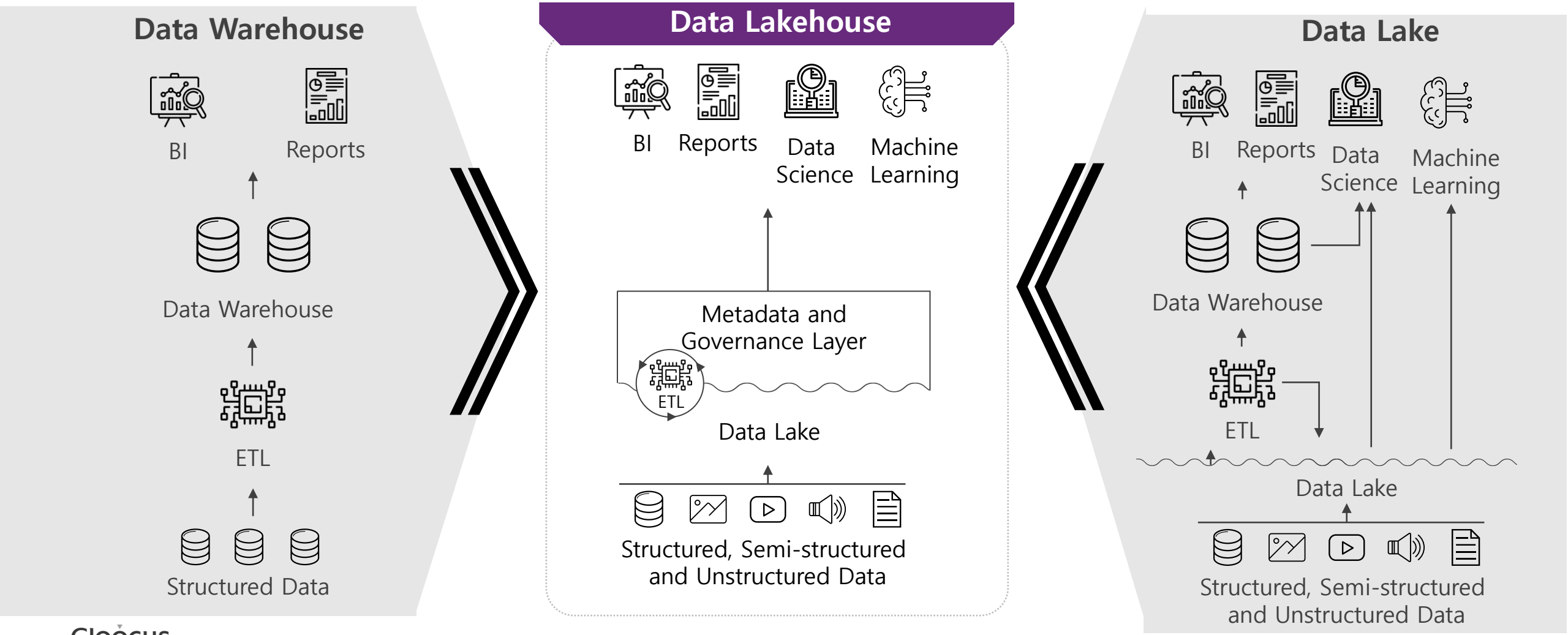
- 데이터 Merge
- 복잡한 데이터 퀄리티 문제
- BI Reports를 위해 DW 엔진 고려

What Is The Next Paradigm?

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

빅데이터 아키텍처 패러다임

1. Lake House Architecture



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Databricks - Lakehouse 소개

The Data and AI Company

Databricks Korea
임상배/sangbae.lim@databricks.com
Sr. Solutions Architect

databricks

databricks

The Data and AI Company

Lakehouse

모든 데이터, 분석, AI 워크로드를 통합하는
하나의 Simple Platform

전 세계 7000+ 고객과 450+ 파트너 보유

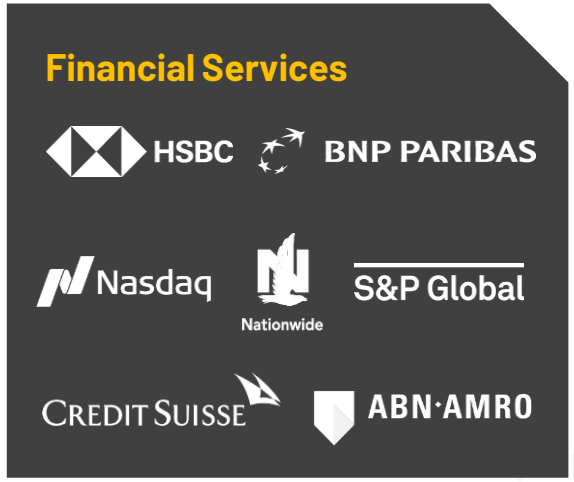
7000+

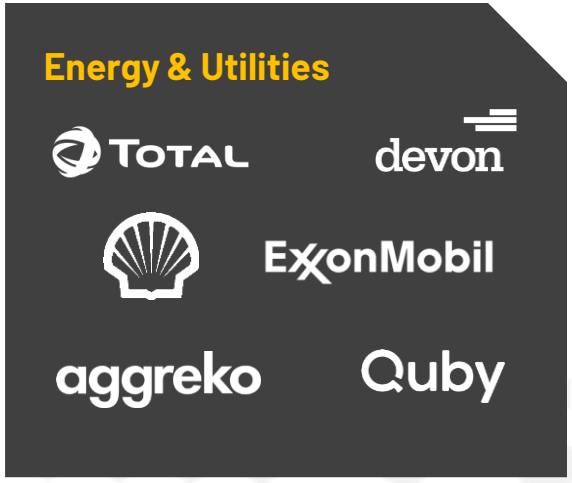
Across the globe

가장 성공적인 데이터 및 머신 러닝 오픈 소스 프로젝트의 오리지널 크리에이터

APACHE Spark™ DELTA LAKE mlflow™ Koalas

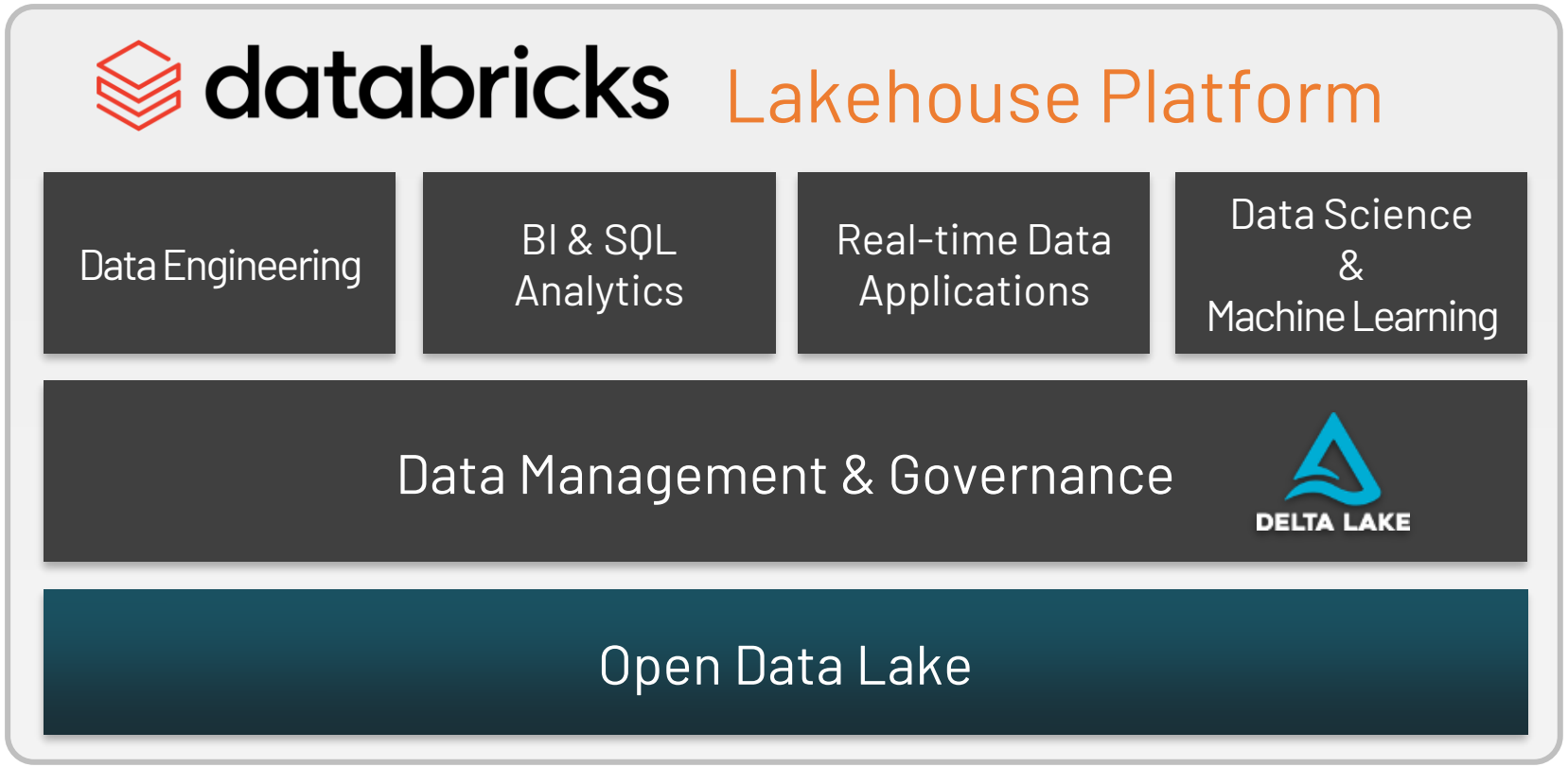
모든 산업군에서 다양한 고객 사례 보유





Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

심플한 데이터 아키텍처로 데이터 사일로 제거



SIMPLE

모든 데이터 활용 사례에 하나의 공통 플랫폼에서 데이터, 분석 및 AI를 통합

OPEN

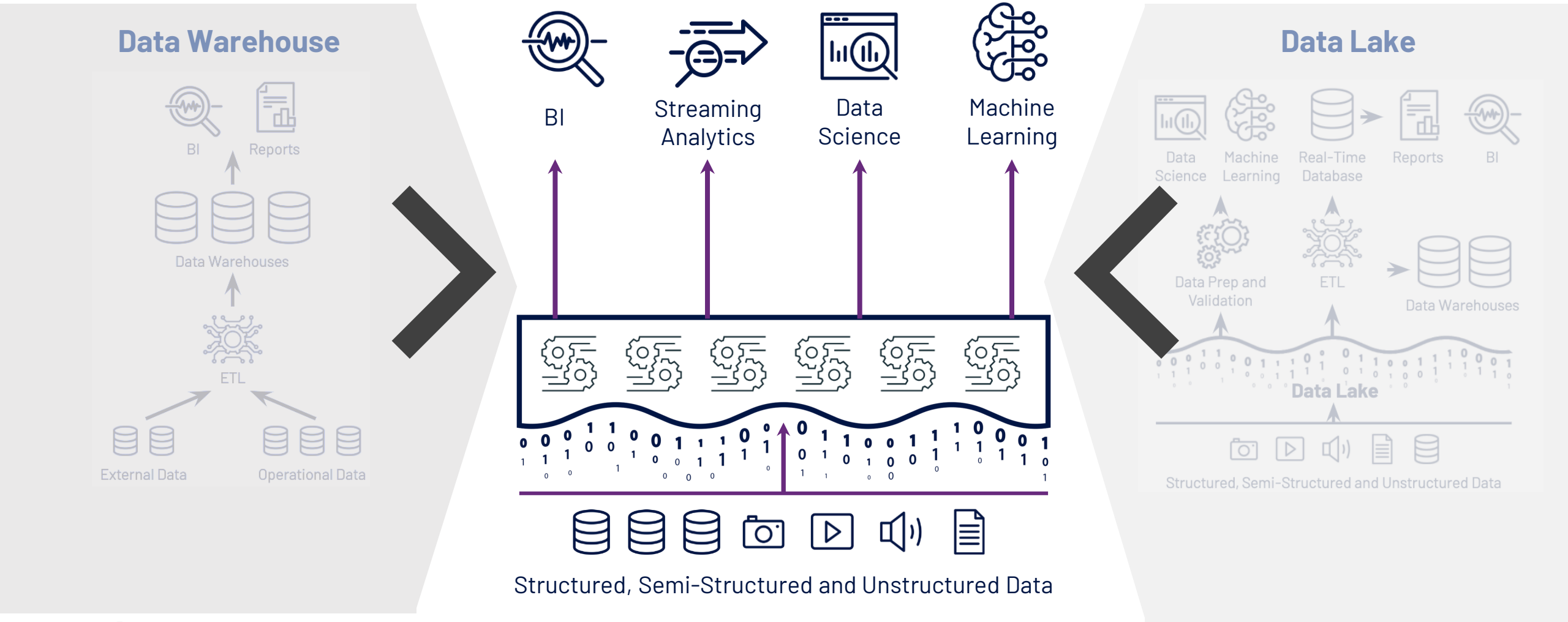
전세계에서 가장 활발하고 성공적인 오픈소스 데이터 프로젝트에 기반한 혁신

COLLABORATIVE

전체적인 데이터와 AI 분석 워크 플로우 전반에 걸친 데이터 팀간의 협업 환경 통합

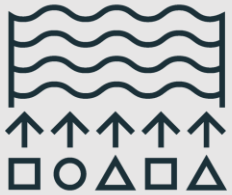
Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Lakehouse는 DW, Data Lake의 장점을 제공



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Data Lake



Lakehouse

모든 데이터, 분석 및 AI 업무를
통합하는 단 하나의 플랫폼

DELTA LAKE

데이터 관리 및 거버넌스를
데이터 레이크에 제공하는
개방형 접근 방법

트랜잭션으로 더 나은 안정성

인덱싱으로 48x 더 빠른 데이터 처리

**세분화된 접근 제어 리스트로
대규모 데이터 거버넌스**

Data Warehouse

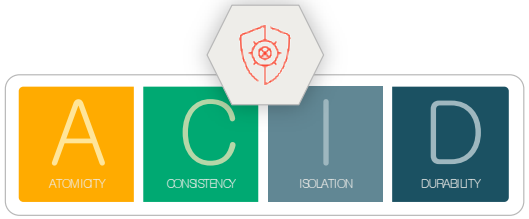
.OOCUS I All Rights reserved

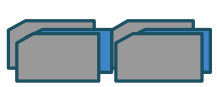
Delta Lake

데이터 레이크 상에 오픈 형식의 저장 레이어 : 100% Apache Spark 호환

데이터 고품질,신뢰성 확보

- ACID 트랜잭션 지원
- Batch + Stream 통합
- Schema Enforcement
- Time Travel





Parquet Files

Transactional Log

성능 최적화

- Indexing
- Z-Ordering / Data Skipping
- Compaction
- Delta Cache
- 24x 빠른 쿼리 속도
  48x 빠른 ETL 처리

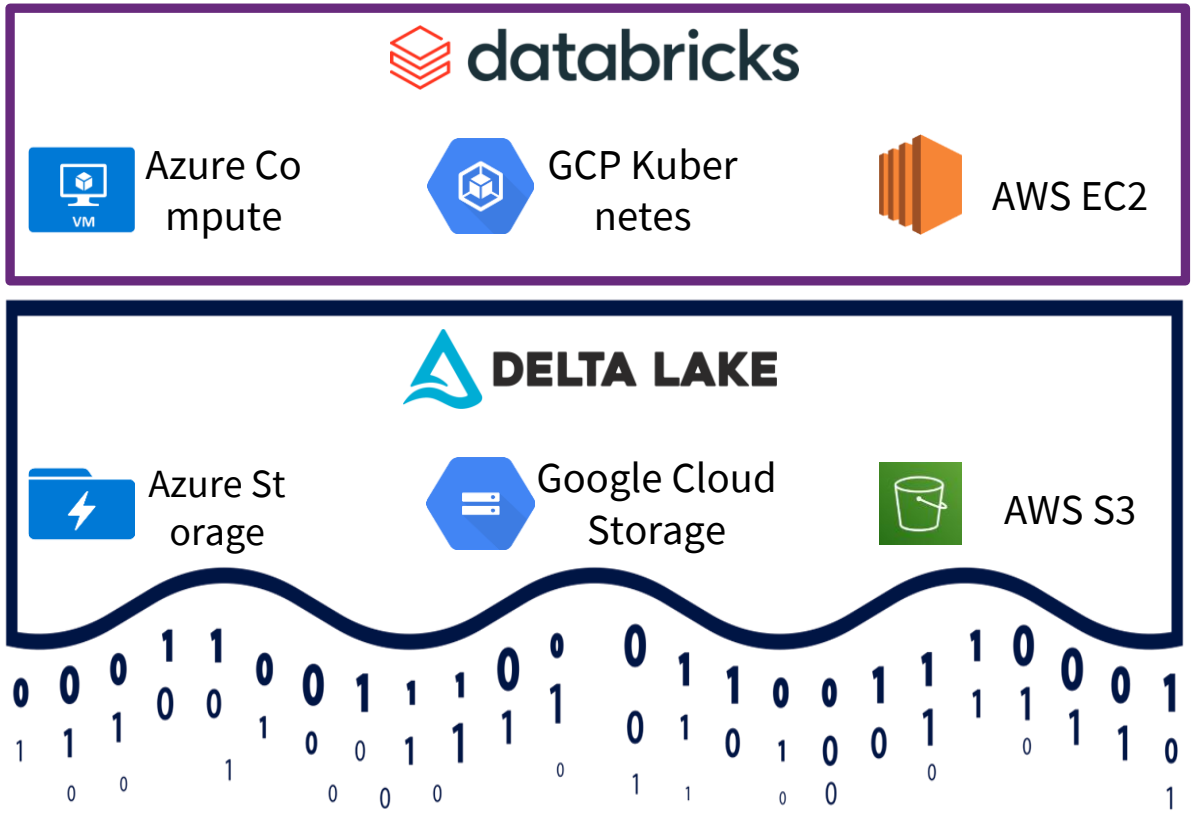
개방 + 보안

- Open Format 활용 타 솔루션에서 접근 가능
- No Vendor Lock-in
- Delta Sharing
- Fine Grained ACL

Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

클라우드 중립적(Agnostic), 멀티 클라우드 지원

- 멀티 클라우드 배포
- 각 클라우드(AWS, Azure, GCP)의 컴퓨팅 기능을 활용
- 모든 클라우드 스토리지에서 하나의 개방형 데이터 형식

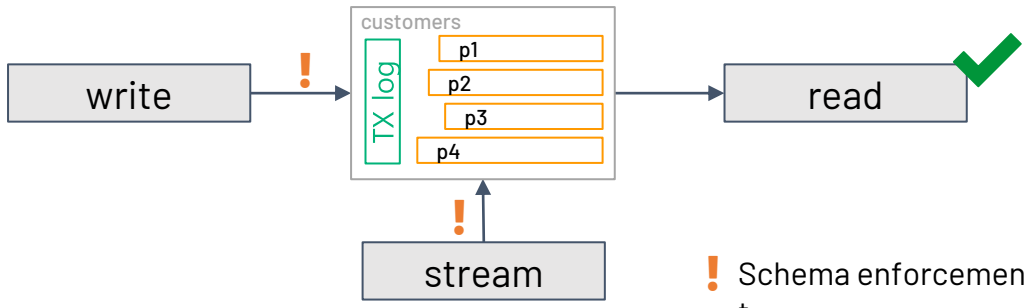


Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Delta에서 신뢰성 및 성능 제공 방안

DELTA LAKE

일관성 보장(Schema validation, Data Quality)



- 배치 데이터 혹은 스트리밍 데이터를 저장소에 쓰기 작업 수행 시, 내장형 Schema enforcement 사용하여 문제가 있는(broken, unfinished, wrong) 데이터를 읽는 것을 방지
- 주어진 Schema만 허용하도록 데이터를 수집하는 것을 쉽게 구성할 수 있으며, 예를 들어 특정 범위 내의 값만 허용하거나 NULL 값을 제거하도록 구성하는 것을 지원

```
( new_health_tracker_data_df.write
    .format("delta")
     .mode("append")
    .save(schema_dir) )
```

```
AnalysisException: A schema mismatch detected when writing to the Delta table (Table ID: 94ea1294-9d88-43
76-bc35-0727970a4280).
```

```
Table schema:
root
-- device_id: long (nullable = true)
-- heartrate: double (nullable = true)
-- name: string (nullable = true)
-- time: double (nullable = true)


Data schema:
root
-- brand: string (nullable = true)
-- device_id: long (nullable = true)
-- heartrate: double (nullable = true)
-- name: string (nullable = true)
-- time: double (nullable = true)
```


Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

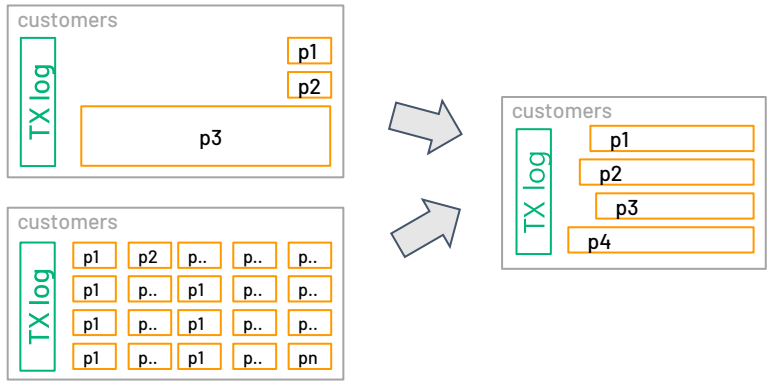
Delta에서 신뢰성 및 성능 제공 방안

DELTA LAKE

쓰기 최적화, Update/Delete 연산 지원

Optimizations on the fly

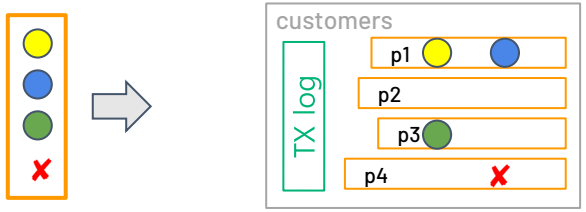
(no need to have a complex pipeline)



- Delta는 복잡한 쓰기 파이프 라인을 구성없이 유입되는 데이터 최적화를 지원하는 지능형 기술
- 데이터 유입 파이프 라인에서 다수의 작은 파일을 쓰거나 혹은 심각하게 치우친 데이터(Data Skew 발생)를 쓸 때 Delta는 자동으로 해당 파일을 병합하여 적당한 크기의 고르게 분포된 파일을 생성
- 일반적인 Parquet 파일에서 발생하는 Small file, Big file 문제를 해결하여 성능을 개선

Direct updates and deletes

(no need to have a complex pipeline)



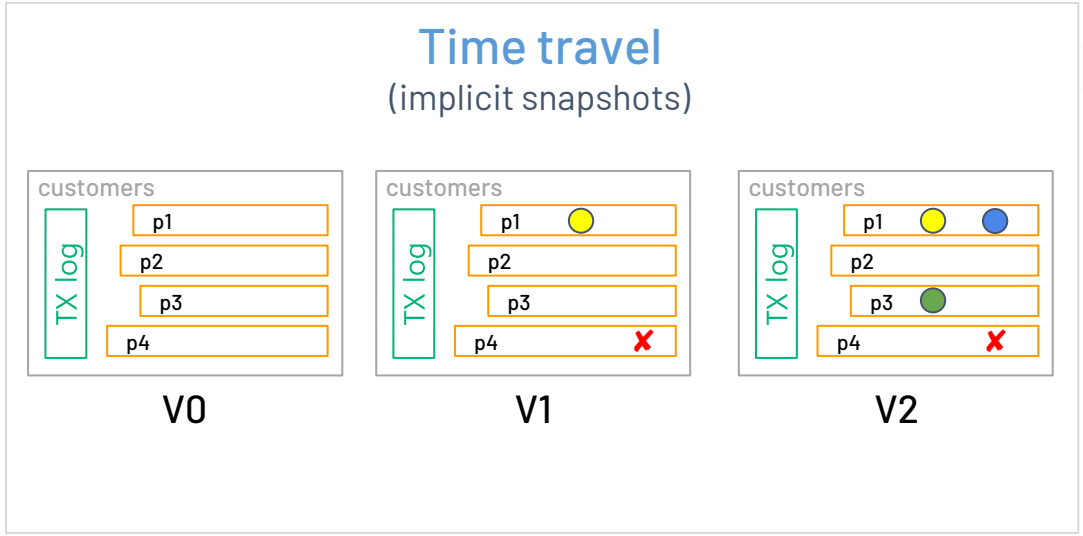
- GDPR
- Change Data Capture (CDC & SCD)

- 일반적인 Parquet 파일은 Append Only이나, Delta는 Transaction Log를 사용하여 update/delete 연산을 지원하여 복잡한 데이터 파이프 라인을 구축하는 것을 방지
- 규제 표준을 충족 및 CDC(Change Data Capture) 요건 충족하는 것을 지원

Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Delta에서 신뢰성 및 성능 제공 방안

Time Travel 기능 제공(원하는 Commit 시점의 데이터를 제공)



테이블 히스토리 확인

```
display(processedDeltaTable.history())
```

(1) Spark Jobs

	version	timestamp	userId	userName	operation	operationParameters
1	3	2021-08-30T16:09:41.000+0000	1728243541534360	sangbae.lim@databricks.com	MERGE	{"predicate": "((health_tracker.`time` = upser "matchedPredicates": "[{\"actionType\":\"upd
2	2	2021-08-30T15:54:51.000+0000	1728243541534360	sangbae.lim@databricks.com	WRITE	{"mode": "Append", "partitionBy": "[]"}
3	1	2021-08-30T14:59:04.000+0000	1728243541534360	sangbae.lim@databricks.com	REPLACE COLUMNS	{"columns": "[{\"name\":\"dte\",\"type\":\"da {\"name\":\"time\",\"type\":\"timestamp\",\"nu {\"name\":\"heartrate\",\"type\":\"double\",\"n {\"name\":\"name\",\"type\":\"string\",\"nullab {\"name\":\"p_device_id\",\"type\":\"integer\",
4	0	2021-08-30T14:49:25.000+0000	1728243541534360	sangbae.lim@databricks.com	CONVERT	{"numFiles": "5", "partitionedBy": "[\"p_dev

versionAsOf, 특정 버전으로 Time Travel

- Transaction log를 사용해서 버전 혹은 타임스탬프로 사용자 데이터의 스냅샷을 가져오는 것을 지원
- 데이터프레임에서 .history() 호출하여 버전, 타임스탬프, 사용자 정보, 연산, 연산 파라미터 정보 확인

```
(spark.read
  .option("versionAsOf", 2)
  .format("delta")
  .load(health_tracker + "processed")
  .count())
```

(5) Spark Jobs

```
Out[17]: 7128
```

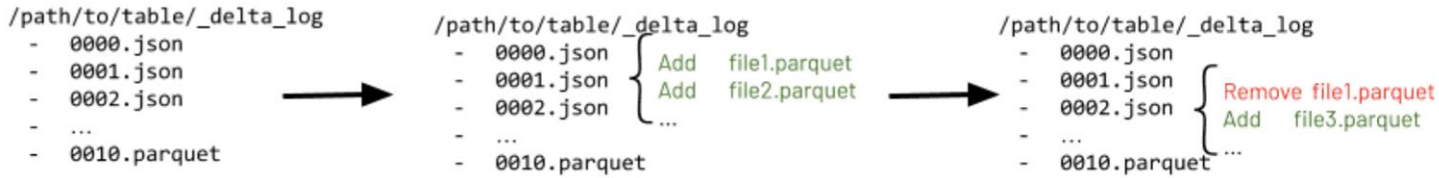
```
# ANSWER
from pyspark.sql.functions import lit

upsertsDF = (
  spark.read
  .option("versionAsOf", 5)
  .format("delta")
  .load(health_tracker + "processed")
  .where("p_device_id = 4")
  .select("dte", "time",
          "heartrate", lit(None).alias("name"), "p_device_id")
)
```

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Delta Table 개요

Parquet 파일 기반으로, Transaction log를 지원

- Delta Table은 Delta Lake 기술을 사용하여 유지하는 데이터 모음으로 3 가지 항목으로 구성됨
- Delta File(Apache Parquet 파일 사용), Transaction Log, Metastore에 등록된 테이블(optional)
- Delta Table 정의 방법 :
  - CREATE TABLE example_table USING **PARQUET**
  - CREATE TABLE example_table USING **DELTA**

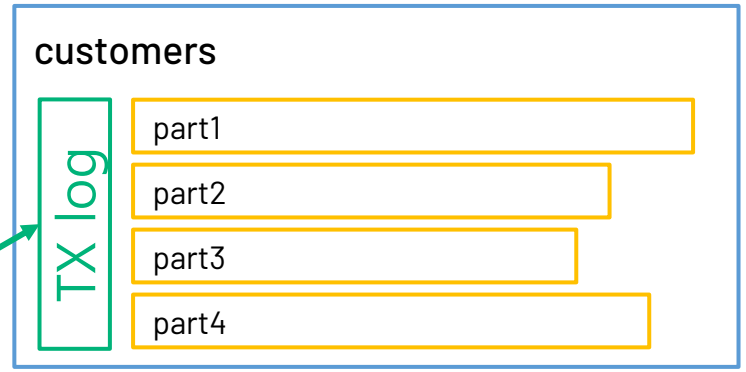


Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Delta Table 개요

Transaction Log Provides Metadata Layer for Consistency

Data in Parquet




```
%fs ls /tmp/bernhard/loan_by_state_delta
```

path
dbfs:/tmp/bernhard/loan_by_state_delta/_delta_log/
dbfs:/tmp/bernhard/loan_by_state_delta/part-00000-cd80fd12-457b-4058-a904-3628c6e90a57-c000.snappy.parquet
dbfs:/tmp/bernhard/loan_by_state_delta/part-00004-69b2735a-18d1-474e-8318-ecd13ca410a7-c000.snappy.parquet
dbfs:/tmp/bernhard/loan_by_state_delta/part-00009-05545d85-f051-4a37-852a-18298fb4f3cd-c000.snappy.parquet
dbfs:/tmp/bernhard/loan_by_state_delta/part-00010-3fa88ba6-61cc-45cd-8c0d-e0949d1bbcce-c000.snappy.parquet

https://databricks.com/blog/2019/08/21/diving-into-delta-lake-unpacking-the-transaction-log.html

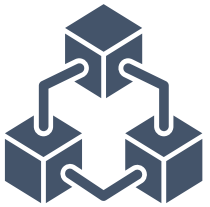
Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Spark APIs를 사용해서 Delta로 손쉽게 변환

parquet... 대신에

```
CREATE TABLE ...
USING parquet
...

dataframe
    .write
    .format("parquet")
    .save("/data")
```

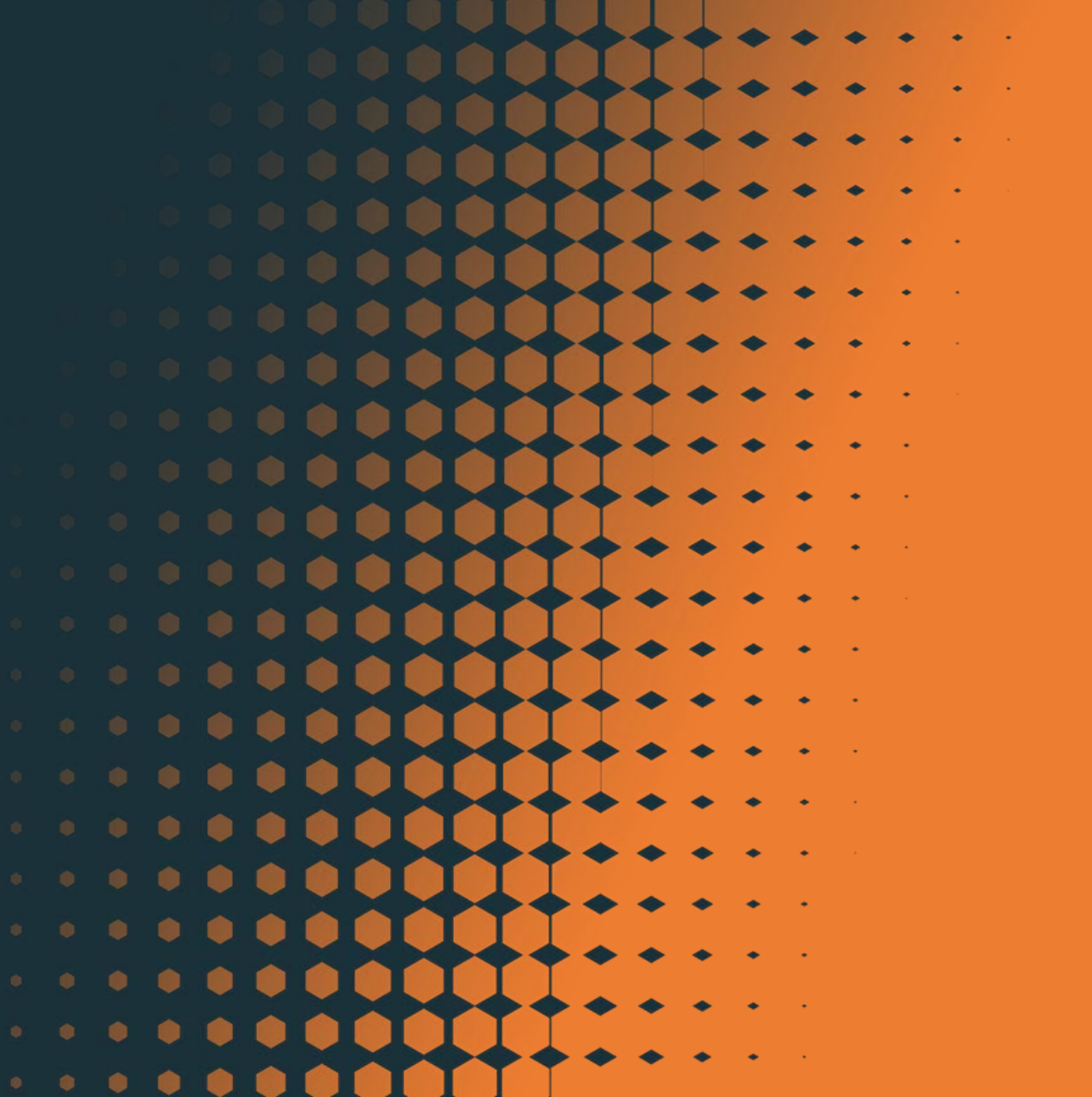
Delta로 치환



```
CREATE TABLE ...
USING delta
…

dataframe
    .write
    .format("delta")
    .save("/data")
```

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Delta Lake Demo



databricks

DeltaLake_Demo Python

sangbae-demo | File | Edit | View: Standard | Permissions | Run All | Clear

데이터 파일 위치

lakedemo

Cmd 5

1.Delta Lake에서 데이터 처리 방법 : Bronze/Silver/Gold

- 브론즈 : raw data 그대로 저장
- 실버 : 데이터 클린징 및 강화해서 "Single version of truth" 데이터 생성
- 골드 : 특정 목적으로 집계/재구성한 데이터(데이터마트)

원시 데이터를 Dataframe으로 읽고 SQL 뷰를 생성한 후 표준 SQL로 데이터 처리

Cmd 6

```
%fs
ls /FileStore/flight/lakedemo
```

```
%fs
ls /FileStore/flight/lakedemo
```

	path	name	size
1	dbfs:/FileStore/flight/lakedemo/assignment_1_backfill.csv	assignment_1_backfill.csv	12890
2	dbfs:/FileStore/flight/lakedemo/assignment_1_ingest.csv	assignment_1_ingest.csv	771013069

Showing all 2 rows.

Command took 5.33 seconds -- by sangbae.lim@databricks.com at 12/11/2021, 11:21:10 on sangbae-demo

CSV 형식의 파일

Cmd 7

Python

```
# IoT 히스토리컬 데이터를 읽어서 데이터프레임으로 처리
dataPath = f"dbfs:/FileStore/flight/{rawdata_location}/assignment_1_ingest.csv"
df = spark.read.option("header","true").option("inferSchema","true").csv(dataPath)
display(df)
```

원본 CSV 파일을 읽기

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [id: string, reading_time: timestamp ... 6 more fields]

	id	reading_time	device_type	device_id	device_operational_status	reading_1	reading_2	reading_3
1	34fb2d8a-5829-4036-adea-a08ccc2c260c	2015-02-23T10:18:54.958+0000	RECTIFIER	7G007R	NOMINAL	19.9996185303	0.03656006	0.030136108
2	f7caf2f0-773e-4e1f-a530-d1f5342193c2	2015-02-23T10:18:54.964+0000	RECTIFIER	7G007R	NOMINAL	19.998550415	0.035491943	0.027999878
3	597a91bc-6bcd-492c-9dd7-3b3a9dabf911	2015-02-23T10:18:54.968+0000	RECTIFIER	7G007R	NOMINAL	20.0006866455	0.033355713	0.030136108
4	21801953-c2ff-4335-98a4-75f9000248f3	2015-02-23T10:18:54.976+0000	RECTIFIER	7G007R	NOMINAL	20.0006866455	0.033355713	0.022659302
5	d1a91019-bd31-4334-b17a-2de9522c8ef4	2015-02-23T10:18:54.978+0000	RECTIFIER	7G007R	NOMINAL	19.9996185303	0.030151367	0.025863647
6	93ed29ce-b7e9-4c5f-b7cd-d85f3d967b39	2015-02-23T10:18:54.992+0000	RECTIFIER	7G007R	NOMINAL	19.9996185303	0.025878906	0.023727417

Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

```
# 과거 데이터 보충(backfill), 나중에 메인 히스토리컬 데이터와 병합
dataPath = f"dbfs:/FileStore/flight/{rawdata_location}/assignment_1_backfill.csv"
df_backfill = spark.read.option("header","true").option("inferSchema","true").csv(dataPath)

display(df_backfill)
```

보충 CSV 파일을 읽기

▸ (3) Spark Jobs

▸ df_backfill: pyspark.sql.dataframe.DataFrame = [id: string, reading_time: timestamp ... 6 more fields]

	id	reading_time	device_type	device_id	device_operational_status	reading_1	reading_2	readi
1	34fb2d8a-5829-4036-adea-a08ccc2c260c	2015-02-23T10:18:54.958+0000	RECTIFIER	7G007R	NOMINAL	19.9996185313	0.03656006	0.030
2	f7caf2f0-773e-4e1f-a530-d1f5342193c2	2015-02-23T10:18:54.964+0000	RECTIFIER	7G007R	NOMINAL	19.998550416	0.035491943	0.027
3	597a91bc-6bcd-492c-9dd7-3b3a9dabf911	2015-02-23T10:18:54.968+0000	RECTIFIER	7G007R	NOMINAL	20.0006866465	0.033355713	0.030
4	21801953-c2ff-4335-98a4-75f9000248f3	2015-02-23T10:18:54.976+0000	RECTIFIER	7G007R	NOMINAL	20.0006866465	0.033355713	0.022
5	d1a91019-bd31-4334-b17a-2de9522c8ef4	2015-02-23T10:18:54.978+0000	RECTIFIER	7G007R	NOMINAL	19.9996185313	0.030151367	0.025
6	93ed29ce-b7e9-4c5f-b7cd-d85f3d967b39	2015-02-23T10:18:54.992+0000	RECTIFIER	7G007R	NOMINAL	19.9996185313	0.025878906	0.023

Showing all 100 rows.

Command took 1.54 seconds -- by sangbae.lim@databricks.com at 12/11/2021, 11:21:10 on sangbae-demo

Cmd 9

Python

```
#SQL로 처리를 위해서 데이터프레임 상에 임시뷰를 생성

df.createOrReplaceTempView("historical_bronze_vw")
df_backfill.createOrReplaceTempView("historical_bronze_backfill_vw")
```

SQL 처리를 위해 임시 뷰 생성

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Bronze 테이블 생성

```
%sql
-- Delta Lake 테이블 생성
DROP TABLE IF EXISTS sensor_readings_historical_bronze;

CREATE TABLE sensor_readings_historical_bronze USING DELTA
as select * from historical_bronze_vw
```

```
%sql
--[624만건 데이터]
SELECT COUNT(*) FROM sensor_readings_historical_bronze
```

▸ (2) Spark Jobs

	count(1)
1	6240991

2.브론즈 테이블은 문제점이 있어 이를 해결한 실버 테이블이 필요

- backfill 데이터가 있어 적용해야 함
- 다양한 오류 발생 시 999.99 값이 전송됨

Cmd 17

Silver 테이블 생성 (데이터 정제, 변환)

```
%sql
-- 브론즈 테이블은 몇 가지 문제가 있어 이를 보완하여 실버 테이블 생성

DROP TABLE IF EXISTS sensor_readings_historical_silver;

CREATE TABLE sensor_readings_historical_silver USING DELTA
as select * from historical_bronze_vw;
```

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

```
%sql
-- 블론즈 테이블에 보충 데이터 테이블을 병합
-- Insert/Update 2개를 하나의 배치에서 수행
-- Delta Lake를 사용하면 기존 Append Only 오브젝트 스토리지는 Update가 가능한 저장소로, 배치와 실시간을 모두 지원하는 고도화된 객체 저장소로 진화됨

MERGE INTO sensor_readings_historical_silver AS silver -- TO DO... "AS" what?
USING historical_bronze_backfill_vw AS bronze-- TO DO... "AS" what?
ON  silver.id=bronze.id
WHEN MATCHED THEN update set *
WHEN NOT MATCHED THEN insert *;
```

원본 파일 + 보충 파일 적용
(신규, 및 변경)

▸ (9) Spark Jobs

	num_affected_rows	num_updated_rows	num_deleted_rows	num_inserted_rows
1	100	54	0	46

Showing all 1 rows.

동일 키 값 업데이트

신규 데이터 추가

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

```sql
%sql
-- [문제가 있는 데이터를 받으면 데이터 리딩파트에서 999.99를 전송하도록 되어 있었음]
SELECT *
FROM sensor_readings_historical_silver
WHERE reading_1 = 999.99
```

문제 있는 데이터 확인 (999.99)

▸ (3) Spark Jobs

	id	reading_time	device_type	device_id	device_operational_status	reading_1	reading_2	reading_3
1	80ffbdd8-65d4-4b7d-b4d3-f02					999.99	999.99	999.99
2	ee2aead8-55c6-4357-ae03-ac					999.99	999.99	999.99
3	b6a30210-b823-4c31-ab62-2c					999.99	999.99	999.99
4	7d67b4ae-79d7-459e-848e-67					999.99	999.99	999.99
5	bdacc82b-adf2-4a85-9469-da					999.99	999.99	999.99
6	1922bf1e-ba54-4370-ac81-40					999.99	999.99	999.99

Showing all 367 rows.

```sql
%sql
-- SQL 윈도우 함수를 사용해서 999.99로 수신된 데이터는 앞/뒤 데이터의 평균값으로 보정할 필요가 있음(보간법)

DROP TABLE IF EXISTS sensor_readings_historical_interpolations;

CREATE TABLE sensor_readings_historical_interpolations AS (
  WITH lags_and_leads AS (
    SELECT
      id,
      reading_time,
      device_type,
      device_id,
      device_operational_status,
      reading_1,
      LAG(reading_1, 1, 0)  OVER (PARTITION BY device_id ORDER BY reading_time ASC, id ASC) AS reading_1_lag,
      LEAD(reading_1, 1, 0) OVER (PARTITION BY device_id ORDER BY reading_time ASC, id ASC) AS reading_1_lead,
      reading_2,
      LAG(reading_2, 1, 0)  OVER (PARTITION BY device_id ORDER BY reading_time ASC, id ASC) AS reading_2_lag,
      LEAD(reading_2, 1, 0) OVER (PARTITION BY device_id ORDER BY reading_time ASC, id ASC) AS reading_2_lead,
      reading_3,
      LAG(reading_3, 1, 0)  OVER (PARTITION BY device_id ORDER BY reading_time ASC, id ASC) AS reading_3_lag,
      LEAD(reading_3, 1, 0) OVER (PARTITION BY device_id ORDER BY reading_time ASC, id ASC) AS reading_3_lead
    FROM sensor_readings_historical_silver
  )
  SELECT
    id,
    reading_time,
    device_type,
    device_id,
    device_operational_status,
    ((reading_1_lag + reading_1_lead) / 2) AS reading_1,
    ((reading_2_lag + reading_2_lead) / 2) AS reading_2,
    ((reading_3_lag + reading_3_lead) / 2) AS reading_3
  FROM lags_and_leads
  WHERE reading_1 = 999.99
  ORDER BY id ASC
)
```

보간법으로 오류 데이터 보완(LAG, LEAD)

```sql
%sql
-- 의도한 방식으로 센서 수치가 변경되었음을 확인

SELECT * FROM sensor_readings_historical_interpolations
```

센서 값 보정 확인

▸ (1) Spark Jobs

	id	reading_time	device_type	device_id	device_operational_status	reading_1	reading_2
1	006d540e-867b-4965-94d7-e170fc283fd0	2015-02-24T14:41:01.514+0000	RECTIFIER	5E005R	RESETTING	49.736801154999995	-0.8395538499999999
2	0124f6d5-7582-48c7-b741-fc9a876149d4	2015-02-23T10:28:39.744+0000	RECTIFIER	7G007R	IDLE	9.999618530300001	0.0034790039
3	037b2a16-3cdf-4b09-a9f4-912e42b37706	2015-02-24T14:19:15.858+0000	RECTIFIER	5E005R	IDLE	10.0062866211	-0.0160980225
4	0386a982-c5da-480a-b0b9-096f376ba2d1	2015-02-24T11:25:22.647+0000	TRANSFORMER	9I009T	RISING	9.985046385	0.46627808
5	03f48ac3-b0b8-439e-852d-89a64ff094fd	2015-02-24T14:20:41.784+0000	RECTIFIER	5E005R	IDLE	9.9946594237	0.0057678223
6	041a324c-7bd2-4117-975d-bd93f0fed50c	2015-02-24T12:03:25.872+0000	RECTIFIER	6F006R	IDLE	9.959762573999999	-0.0402984625

```sql
%sql
-- [문제가 있는 데이터를 받으면 데이터 리딩파트에서 999.99를 전송하도록 되어 있었음]
SELECT *
FROM sensor_readings_historical_silver
WHERE reading_1 = 999.99
```

▸ (3) Spark Jobs

	id	reading_time	device_type	device_id	device_operational_status	reading_1	reading_2	reading_3
1	80ffbdd8-65d4-4b7d-b4d3-f02aef04839b	2015-02-24T13:10:09.625+0000	TRANSFORMER	4D004T	HIGH	999.99	999.99	999.99
2	ee2aead8-55c6-4357-ae03-ac25f5510cff	2015-02-24T13:10:52.444+0000	TRANSFORMER	4D004T	HIGH	999.99	999.99	999.99
3	b6a30210-b823-4c31-ab62-2d81934a1527	2015-02-24T13:30:51.108+0000	TRANSFORMER	2B002T	NOMINAL	999.99	999.99	999.99
4	7d67b4ae-79d7-459e-848e-67c758e5fa9b	2015-02-24T13:31:56.002+0000	RECTIFIER	2B002R	NOMINAL	999.99	999.99	999.99
5	bdacc82b-adf2-4a85-9469-dad73ffc0c44	2015-02-24T13:32:46.088+0000	TRANSFORMER	2B002T	NOMINAL	999.99	999.99	999.99
6	1922bf1e-ba54-4370-ac81-409c5a12dc42	2015-02-24T13:33:46.128+0000	TRANSFORMER	2B002T	NOMINAL	999.99	999.99	999.99

Showing all 367 rows.

```sql
%sql
-- 뒤늦게 수신된 데이터로 데이터 보완 => 오류 데이터(999.99)를 보간법으로 보정
-- Merge 문을 사용해서 실버 테이블의 데이터 더욱 의미있게 만들기
merge into sensor_readings_historical_silver as S
using sensor_readings_historical_interpolations as I
on S.id=I.id
when matched then update set *
when not matched then insert *;
```

MERGE 문으로 데이터 수정

▸ (9) Spark Jobs

	num_affected
1	367

```sql
%sql
-- 오류 데이터가 남아 있는지 다시 한번 확인
-- [그런데 성능 무엇?(Delta Lake는 통계를 저장 - Delta 트랜잭션 로그에 컬럼의 최소, 최대값이 있어 where 절 평가 시 이를 활용)]

SELECT count(*)
FROM sensor_readings_historical_silver
WHERE reading_1 = 999.99
```

오류 데이터 남아 있는 지 확인

▸ (1) Spark Jobs

	count(1)
1	0

Showing all 1 rows.

Command took 0.23 seconds -- by sangbae.lim@

Command took 0.23 seconds -- by sangbae.lim@databricks.com at 12/11/2021, 12:04:46 on sangbae-demo

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

3.Time Travel - 시간 여행을 해보시죠!

Cmd 26

```
%sql
-- [현재 테이블에서 볼 수 있는 모든 변경 내역을 확인(보유 기간 및 관리는 Admin이 수행)]
DESCRIBE HISTORY sensor_readings_historical_silver
```

Silver 테이블 버전 히스토리 확인

▸ (1) Spark Jobs

	version	timestamp	userId	userName	operation
1	2	2021-11-12T02:41:51.000+0000	1728243541534360	sangbae.lim@databricks.com	MERGE
2	1	2021-11-12T02:38:58.000+0000	1728243541534360	sangbae.lim@databricks.com	MERGE
3	0	2021-11-12T02:38:37.000+0000	1728243541534360	sangbae.lim@databricks.com	CREATE TABLE AS SELECT

Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

오류 센서 값이 있는 데이터 버전 확인

SQL

```
%sql
-- 버전 1 데이터가 999.99 값이 포함되어 있음. 만약 해당 버전의 데이터를 재확인해봐야 한다면?

SELECT *
FROM sensor_readings_historical_silver
version as of 1
WHERE reading_1 = 999.99
```

▸ (3) Spark Jobs

	id	reading_time	device_type	device_id	device_operational_status	reading_1	reading_2	reading_3
1	80ffbdd8-65d4-4b7d-b4d3-f02aef04839b	2015-02-24T13:10:09.625+0000	TRANSFORMER	4D004T	HIGH	999.99	999.99	999.99
2	ee2aead8-55c6-4357-ae03-ac25f5510cff	2015-02-24T13:10:52.444+0000	TRANSFORMER	4D004T	HIGH	999.99	999.99	999.99
3	b6a30210-b823-4c31-ab62-2d81934a1527	2015-02-24T13:30:51.108+0000	TRANSFORMER	2B002T	NOMINAL	999.99	999.99	999.99
4	7d67b4ae-79d7-459e-848e-67c758e5fa9b	2015-02-24T13:31:56.002+0000	RECTIFIER	2B002R	NOMINAL	999.99	999.99	999.99
5	bdacc82b-adf2-4a85-9469-dad73ffc0c44	2015-02-24T13:32:46.088+0000	TRANSFORMER	2B002T	NOMINAL	999.99	999.99	999.99
6	1922bf1e-ba54-4370-ac81-409c5a12dc42	2015-02-24T13:33:46.128+0000	TRANSFORMER	2B002T	NOMINAL	999.99	999.99	999.99

Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

4.분석가/데이터과학자가 필요한 골드 테이블 생성

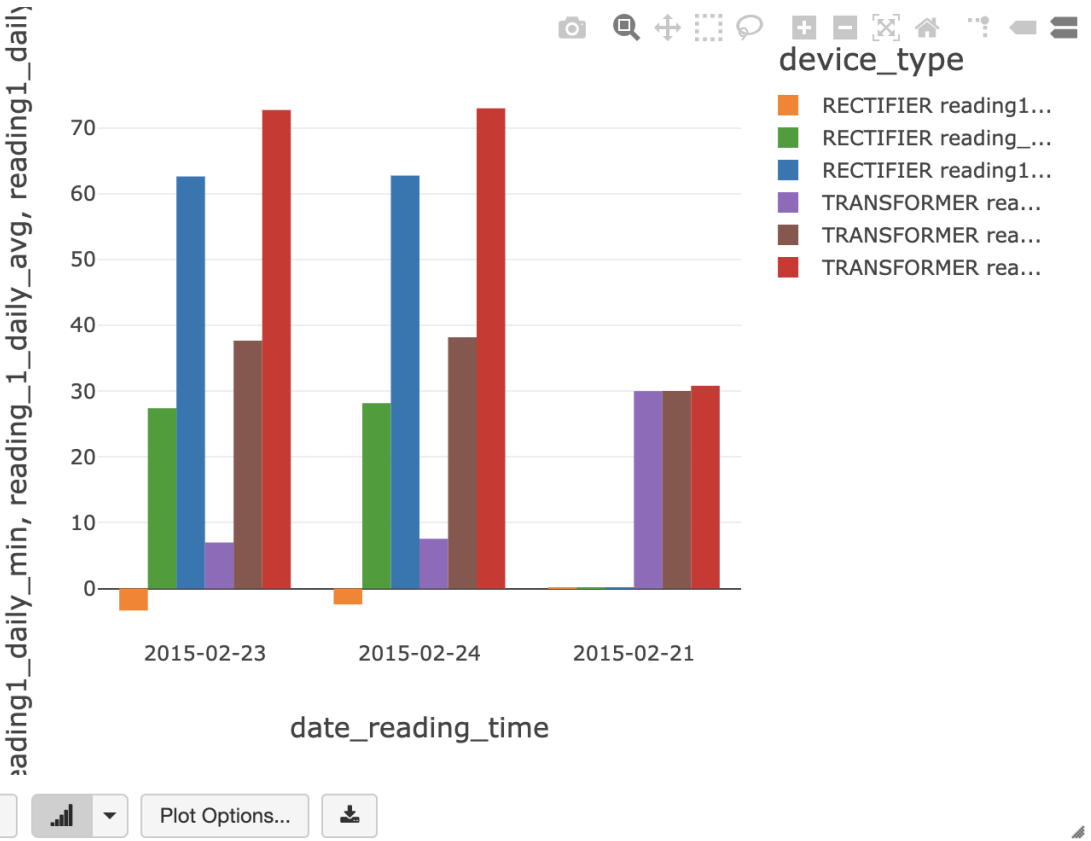
Cmd 41

```
%sql
-- 장비 종류, 날짜 별 센서1번의 평균/최소/최대 수치 집계

drop table if exists sensor_readings_historical_gold_by_reading_1_daily

create table sensor_readings_historical_gold_by_daily_avg_min_max
using delta
as select device_type,date_reading_time,avg(reading_1) as reading_1_dai
from sensor_readings_historical_silver_by_hour_and_minute
group by device_type,date_reading_time;
```

GOLD 테이블 생성(Data Mart)

```
%sql
select * from sensor_readings_historical_gold_by_daily_avg_min_max
order by device_type, date_reading_time
```

▸ (1) Spark Jobs

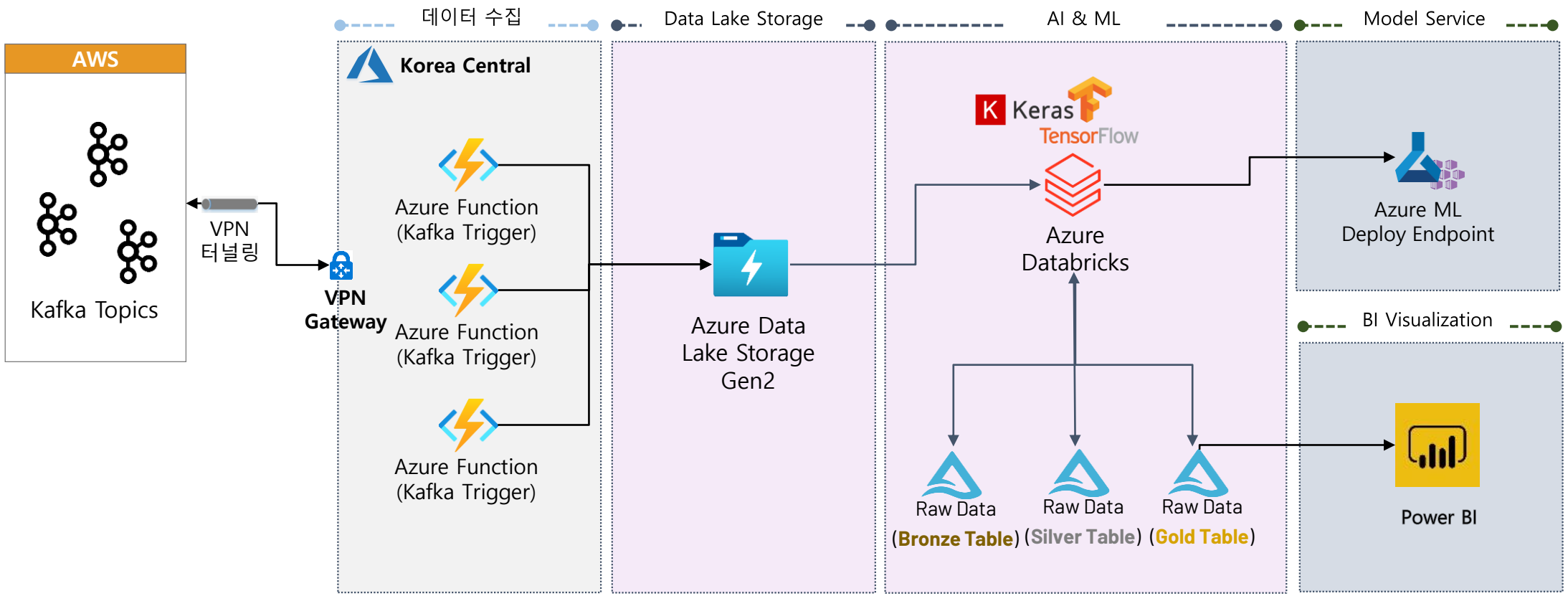


Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Lake House
활용 사례

Lake House 활용

자동차 Industry / H사 Refence – Data Lake

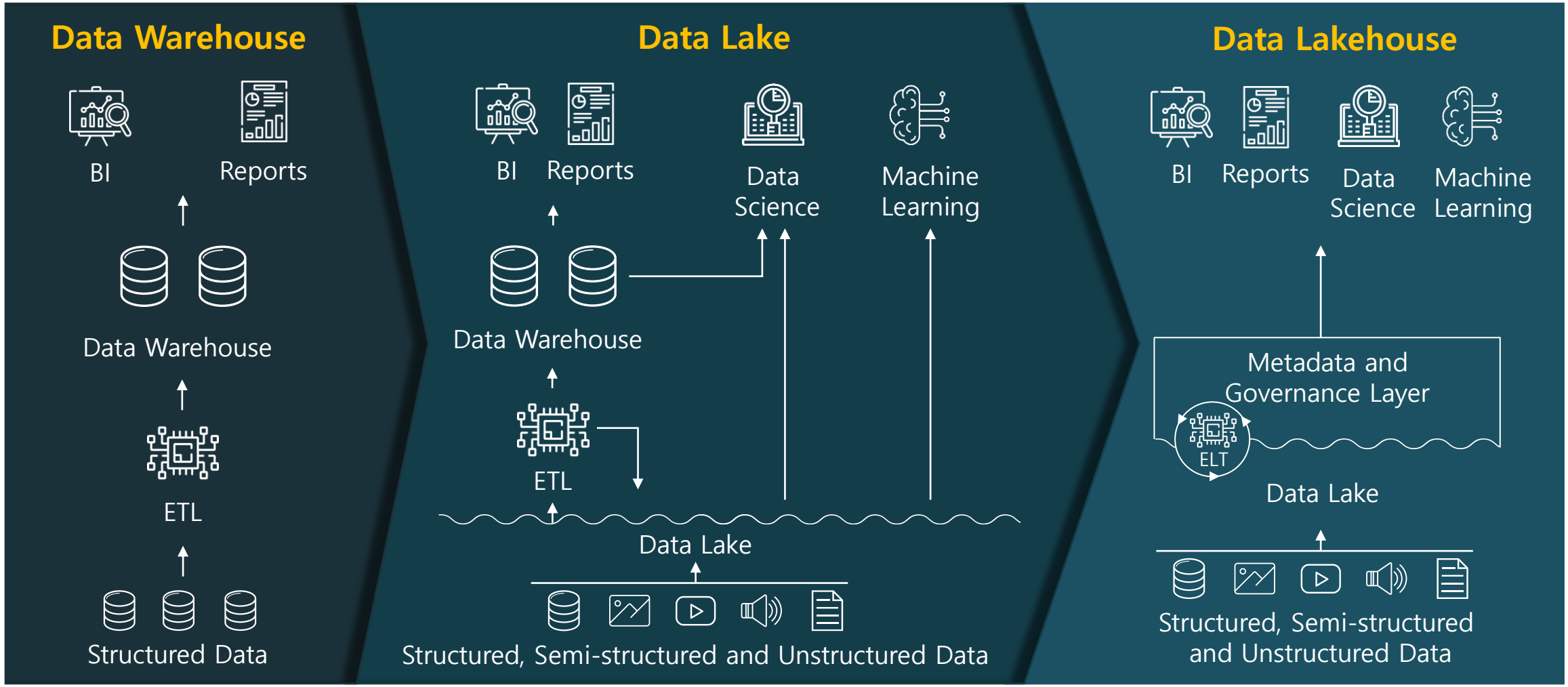


Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

클라우드 데이터,
제대로 활용할 수 있는 3가지
Conclusion

클라우드 데이터, 제대로 활용할 수 있는 3가지

결론



Cloocus Confidential & Proprietary I 2021 CLOOCUS I All Rights reserved

Thank you !

Data Analytics Team,
Data Engineer 박 민지

Focus on Cloud
Microsoft Azure Consulting Expert Group

Cloocus

Gold Microsoft Partner
Microsoft

Azure Expert MSP