



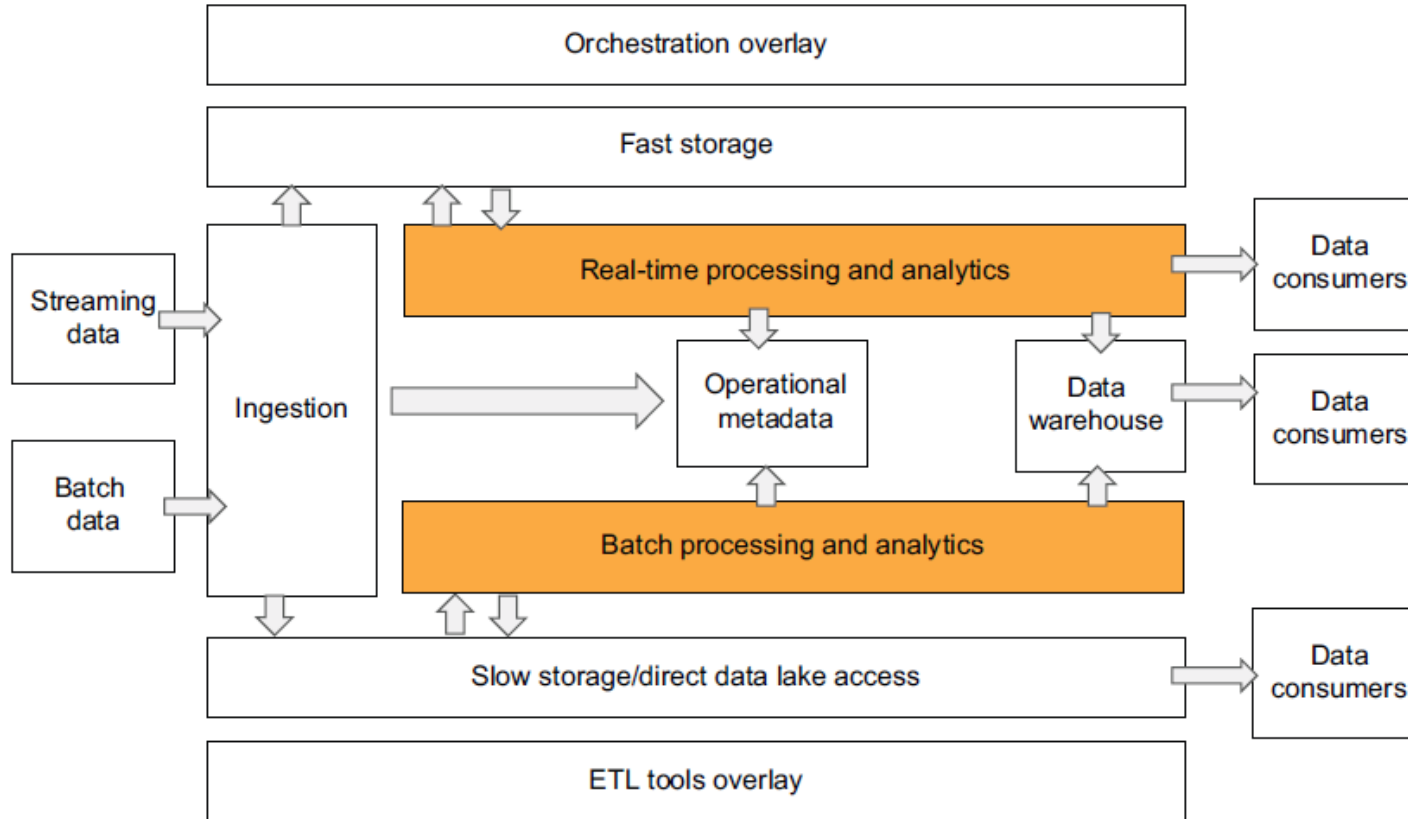
ORGANIZING AND PROCESSING DATA

주요 내용

- 클라우드 데이터 플랫폼에서 데이터 구성 및 처리
- 데이터 처리의 스테이지(stage) 단계 이해
- 컴퓨팅과 스토리지를 분리하는 이유
- 클라우드 스토리지의 데이터 구성 및 데이터 흐름 설계
- 공통 데이터 처리 패턴 구현
- 아카이브, 스테이징, 프로덕션 환경에 적합한 파일 포맷 선택
- 공통 데이터 변환을 사용해 단일 파라미터 기반 파이프라인 생성

처리 계층(Processing Layer)

데이터 유효성 검사, 데이터 변환 작업, 모든 비즈니스 로직이 수행되는 처리 계층은 데이터 플랫폼 구현의 핵심. 처리 계층은 데이터 플랫폼의 데이터를 ad hoc 방식으로 액세스할 때 필요 환경을 제공하는 중요한 역할을 담당.



처리 계층은 저장소에서 데이터를 읽고 변환한 다음 추가 사용을 위해 저장소에 다시 저장하는 역할을 한다. 처리 계층은 느린 데이터 저장소와 빠른 데이터 저장소 모두에서 작동할 수 있어야 한다.

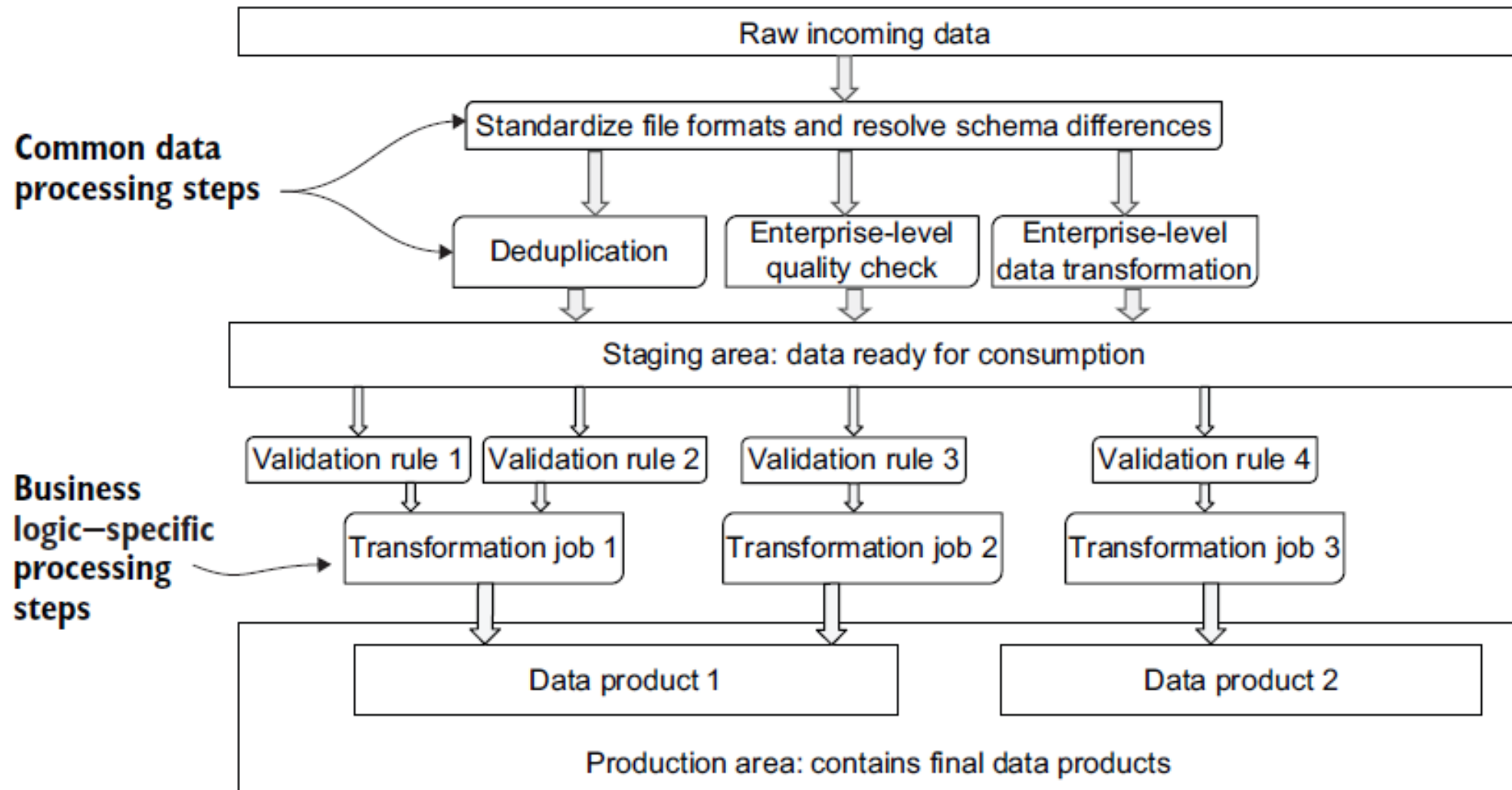
웨어하우스와 레이크에서 데이터 처리

클라우드 데이터 플랫폼의 유연성을 최대한 활용하려 할 경우, 데이터웨어하우스 외부에서 데이터 처리작업을 수행

	데이터레이크에서 데이터 처리(Spark)	데이터웨어하우스에서 데이터 처리(SQL)
유연성	데이터 웨어하우스에서 제공되는 데이터뿐만 아니라 다른 사용자 및 시스템에 전달되거나 소비될 수 있는 데이터에도 출력을 사용할 수 있기 때문에 추가적인 유연성이 제공됨	데이터 처리의 출력은 일반적으로 데이터 웨어하우스에서 사용하도록 제한됨
개발자 생산성	개발자가 교육을 통해 정교한 테스트 프레임워크와 라이브러리를 통해 개발속도를 높일 수 있고, 스파크의 성능과 유연성을 체감할 수 있음	SQL 개발자를 구하는 것이 상대적으로 쉬움
성능	처리량이 증가해도 데이터 웨어하우스 사용자에게 영향을 미치지 않음	처리 부하가 증가할 때 문제가 발생할 수 있음
처리 속도	항상 실시간 분석이 가능	일부 클라우드 데이터웨어하우스에서는 실시간 분석이 가능하지만 여러 단계 또는 제품이 필요함
비용	요금정책이 컴퓨팅 사용량 기반일 경우, 훨씬 저렴	솔루션별 계약조건에 따라 데이터 처리 비용이 많이 들 수 있음

데이터 처리 스테이지

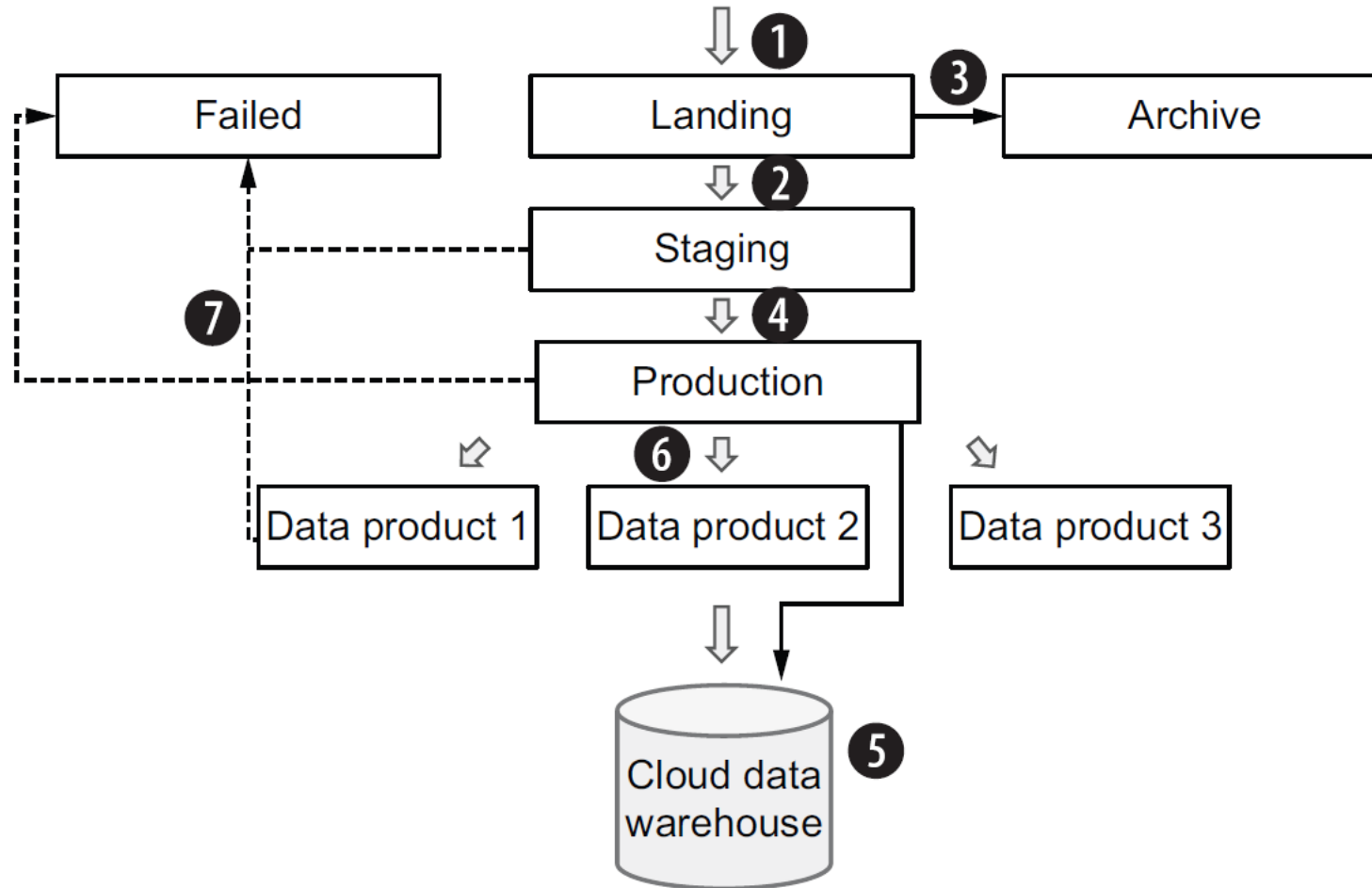
데이터 처리 작업은 일반적으로 공통 데이터 처리 단계와 비즈니스 로직 관련 단계의 두 가지 범주로 나눌 수 있음



클라우드 스토리지 구성

클라우드 스토리지에서 데이터를 구성하는 방법에 대한 일관되고 명확한 원칙을 갖는 것은 매우 중요함.

데이터를 읽을 위치와 데이터 쓰기 위치와 관련하여 동일한 설계를 따르는 표준화된 파이프라인을 구축하면 대규모 파이프라인을 훨씬 쉽게 관리할 수 있고, 스토리지에서 데이터를 검색하고 필요한 데이터를 찾을 위치를 정확히 이해하는 데 도움이 됨.



클라우드 스토리지 구성

1. 수집 계층에서 나온 데이터는 랜딩(Landing) 영역에 저장된다. 랜딩 영역은 원시 데이터가 처리될 때까지 저장돼 있는 곳이다. 랜딩영역에 기록할 수 있는 계층은 수집 계층뿐이다.
2. 원시데이터는 일련의 공통 변환 과정을 거친 다음 스테이징 영역에 저장된다.
3. 원시 데이터는 랜딩영역에서 아카이브(Archive) 영역으로 복제된다. 아카이브의 목적은 재처리가 필요할 경우, 파이프라인 디버깅을 해야 될 경우, 신규 파이프라인 코드를 테스트해야 되는 경우를 대비해 보관한다.
4. 데이터 변환 작업은 스테이징(Staging) 영역에서 데이터를 읽고, 필요한 비즈니스 로직을 적용한 후에 처리 결과를 프로덕션(Production) 영역에 저장한다.
5. 패스 스루(Pass-through) 작업은 선택사항이다. 복제의 목적은 다른 작업의 비즈니스 로직 관련 문제를 디버깅 하는데 필요할 수 있다.
6. 각 단계 작업에서 스테이징 영역으로 부터 데이터를 읽어, 리포팅이나 기타 분석 목적으로 사용할 데이터세트를 생성한다.
7. 흐름의 각 단계별로 장애 시나리오를 준비해야 한다.

클라우드 스토리지 컨테이너와 폴더

클라우드 공급업체는 가격/성능 특성이 다른 다양한 스토리지 계층을 제공함.

핫스토리지 계층은 가장 빠른 읽기/쓰기 작업을 제공하지만 장기적으로 데이터를 저장하는 데 비용이 가장 많이 듦.
콜드 및 아카이브 계층은 속도가 느리지만 핫 계층보다 훨씬 저렴한 비용으로 대용량 데이터를 장기간 저장할 수 있음.

Container	Permissions	Storage tier
Landing	Only ingestion-layer applications are allowed to write to this container. Scheduled pipelines can read data, and data engineers supporting the platform have read/write access. Data consumers don't have access.	Hot. Reads and writes are happening frequently.
Staging	Scheduled pipelines can read/write data, and data engineers supporting the platform have read/write access. Selected data consumers have read-only access.	Hot. Reads and writes are happening frequently.
Production	Scheduled pipelines can read/write data, and data engineers supporting the platform have read/write access. Consumers of Parquet formatted data will have read-only access.	Hot. Reads and writes are happening frequently.
Archive	Scheduled pipelines can write data, and data engineers supporting the platform have read/write access. A dedicated data reprocessing pipeline has read-only access. Very few selected data consumers have read-only access.	Cold or archive. Depending on data volume, you may store your more recent data in the cold archive container and older data in the archive container.
Failed	Scheduled pipelines can write data, and data engineers supporting the platform have read/write access. A dedicated data reprocessing pipeline has read-only access. Data consumers don't have access.	Hot. Reads and writes are happening frequently.

클라우드 스토리지 컨테이너와 폴더

■ 랜딩 컨테이너 폴더 구조

landing/NAMESPACE/PIPELINE/SOURCE_NAME/BATCH_ID/

/landing/ETL/sales_oracle_ingest/customers/01DFTQ028FX89YDFAXREPJTR94

/landing/ETL/sales_oracle_ingest/contracts/01DFTQB596HG2R2CN2QS6EJGBQ

/landing/ETL/marketing_ftp_ingest/campaigns/01DFTQCWAYDPW141VYNMCHSE3

■ 스테이징/프로덕션 컨테이너 폴더구조

/staging/ETL/sales_oracle_ingest/customers/year=2019/month=07/day=03/01DFT

➡ Q028FX89YDFAXREPJTR94

/staging/ETL/sales_oracle_ingest/contracts/year=2019/month=07/day=03/01DFT

➡ QB596HG2R2CN2QS6EJGBQ

/staging/ETL/marketing_ftp_ingest/campaigns/year=2019/month=06/day=01/01D

➡ FTQCWAYDPW141VYNMCHSE3

■ 스트리밍 데이터 구성(아카이빙 목적): UUID 를 고유한 배치 식별자로 사용

/landing/ETL/clickstream_ingest/clicks/01DH3XE2MHJBG6ZF4QKK6RF2Q9










/landing/ETL/clickstream_ingest/clicks/01DH3XFWJVC5K5TDYWATXNDHJ1

/landing/ETL/clickstream_ingest/clicks/01DH3XG81SKYD30YV8EBP82M0K

공통 데이터 프로세싱 단계

파일 포맷을 바이너리 파일 포맷(AVRO 및 Parquet)으로 변환하면 텍스트 기반 파일 포맷을 사용하는 것보다 이점이 많다.
디스크 공간을 훨씬 적게 차지하고, 모든 파일에 대해 특정 스키마 사용을 강제하므로 확장성을 확보하기 훨씬 쉽다.

BIG DATA FORMATS COMPARISON

	Avro	Parquet	ORC
Schema Evolution Support			
Compression			
Splitability			
Most Compatible Platforms	Kafka, Druid	Impala, Arrow Drill, Spark	Hive, Presto
Row or Column	Row	Column	Column
Read or Write	Write	Read	Read

Source: Nexla analysis, April 2018

- orc, parquet, avro는 바이너리 포맷이다.
- orc, parquet, avro는 여러개의 디스크로 나뉘어질수 있으며 이 특징으로 인해 확장성과 동시처리가 가능해진다.
- orc, avro, parquet의 가장 큰 차이점은 데이터를 어떻게 저장 하느냐이다.
- parquet과 orc는 데이터를 column 기반으로 저장하고, avro는 row기반으로 저장한다.
- row기반 저장 → 모든 필드에 접근해야 할때 유용,
- column 기반 저장 → 특정 필드로 자주 접근할 때 유용.
- column기반 저장은 큰 분량을 읽어서 분석해야하는 환경에 최적화되어있고, row기반은 큰 분량을 쓰는데 유리하다.

공통 데이터 프로세싱 단계

■ 스파크를 사용한 파일포맷 변환

```
import datetime
from pyspark.sql import SparkSession
spark = SparkSession.builder ... # we omit Spark session creation for brevity
```

```
→ namespace = "ETL"
  pipeline_name = "click_stream_ingest"
  source_name = "clicks"
  batch_id = "01DH3XE2MHJBG6ZF4QKK6RF2Q9"
  current_date = datetime.datetime.now()
  in_path = f"gs://landing/{namespace}/{pipeline_name}/{source_name}/{batch_id}/*"
  out_path = f"gs://staging/{namespace}/{pipeline_name}/{source_name}/year=
  ➡ {current_date.year}/month={current_date.month}/day={current_date.day}/
  ➡ {batch_id}"

  clicks_df = spark.read.json(in_path)
  clicks_df = spark.write.format("avro").save(out_path)
```

Inputs and outputs GCS paths
following our preferred folder
structure

Spark has native methods for reading
JSON data and inferring its schema.

Saves data in Avro storage
using inferred schema from
the previous step

Defines configuration variables
for our example pipeline

공통 데이터 프로세싱 단계

공통 데이터 처리 단계인 중복 제거에 관한 고유성 강제는 클라우드 데이터 플랫폼에서 특히 중요하다. 일반적으로 신뢰할 수 없는 데이터 소스의 수집을 재실행해야 하는 상황에 대처해야 할 때나, 혹은 기존 클라우드 웨어하우스에서도 데이터의 고유성 적용이 부족한 경우가 있기 때문이다.

■ 데이터 중복 제거

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder ... # we omit Spark session creation for brevity
```

```
namespace = "ETL" ← Defines configuration variables for our example pipeline
```

```
pipeline_name = "users_csv_ingest"
```

```
source_name = "users"
```

```
batch_id = "01DH3XE2MHJBG6ZF4QKK6RF2Q9"
```

```
in_path = f"gs://landing/{namespace}/{pipeline_name}/{source_name}/{batch_id}/*" ←
```

Input and output GCS paths follow our preferred folder structure.

```
users_df = spark.read.format("csv").load(in_path)
```

```
→ users_deduplicate_df = users_df.dropDuplicates(["user_id"])
```

Uses dropDuplicates Spark data frame method to remove rows with duplicate user_id values

Spark has built-in support for reading CSV files.

공통 데이터 프로세싱 단계

■ 조인(Join)을 통해 전역적으로 데이터 중복 제거

```
from pyspark.sql import SparkSession
```

```
spark = SparkSession.builder ... # we omit Spark session creation for brevity
```

```
namespace = "ETL"
```

```
pipeline_name = "users_csv_ingest"
```

```
source_name = "users"
```

```
batch_id = "01DH3XE2MHJBG6ZF4QKK6RF2Q9"
```

```
in_path = f"gs://landing/{namespace}/{pipeline_name}/{source_name}/{batch_id}/*"
```

```
staging_path = f"gs://staging/{namespace}/{pipeline_name}/{source_name}/*"
```

```
incoming_users_df = spark.read.format("csv").load(in_path)
```

```
staging_users_df = spark.read.format("avro").load(staging_path)
```

```
incoming_users_df.createOrReplaceTempView("incoming_users")
```

```
staging_users_df.createOrReplaceTempView("staging_users")
```

```
→ users_deduplicate_df = \
    spark.sql("SELECT * FROM incoming_users u1 LEFT JOIN staging_users u2 ON
    ➡ u1.user_id = u2.user_id WHERE u2.user_id IS NULL")
```

Selects all data from the incoming batch, where a user_id from the incoming batch doesn't occur in the existing data set

We will be reading ALL data from the staging area for this source.

Reads both incoming and existing data in staging

Registers temporary tables for SQL operations

공통 데이터 프로세싱 단계

■ 데이터 품질 검사(예)

- 특정 열에 대한 값의 길이는 사전 정의된 범위 내에 있어야 한다.
- 숫자 값은 적절한 범위 내에 있어야 한다. 예를 들어 음수 급여 값은 허용되지 않는다.
- 특정 열에는 빈 값이 포함되어서는 안 된다.
"비어 있음"이 의미하는 바에 대한 정의는 열마다 다를 수 있다.
- 값은 특정 패턴을 따라야 한다. 예를 들어 이메일 열에는 유효한 이메일 주소가 포함되어야 한다.

Uses the Spark built-in method
to read data from a CSV file

```
users_df = spark.read.format("csv").load(in_path)
bad_user_rows = users_df.filter("length(email) > 100 OR username IS NULL")
users_df = users_df.subtract(bad_user_rows)
```

Creates a new Spark data frame
object by filtering out rows with
emails that are too long or where
the username field is empty

Removes bad rows from the original data
frame by using the Spark subtract method

설정 가능한 파이프라인

데이터 레이크 모델에서 보다 체계화된 데이터 플랫폼 방식으로 전환하면 스토리지에서 데이터 구성방법, 한 스테이지에서 다른 스테이지로 이동하는 방식, 데이터 소비자가 각 스테이지별 데이터를 볼 때의 데이터 형태와 수준 등을 일원화 할 수 있다.

또한, 이러한 체계화 작업을 통해 공통 변환 단계를 고도로 설정 가능한 단일 파이프라인으로 표준화할 수 있다.

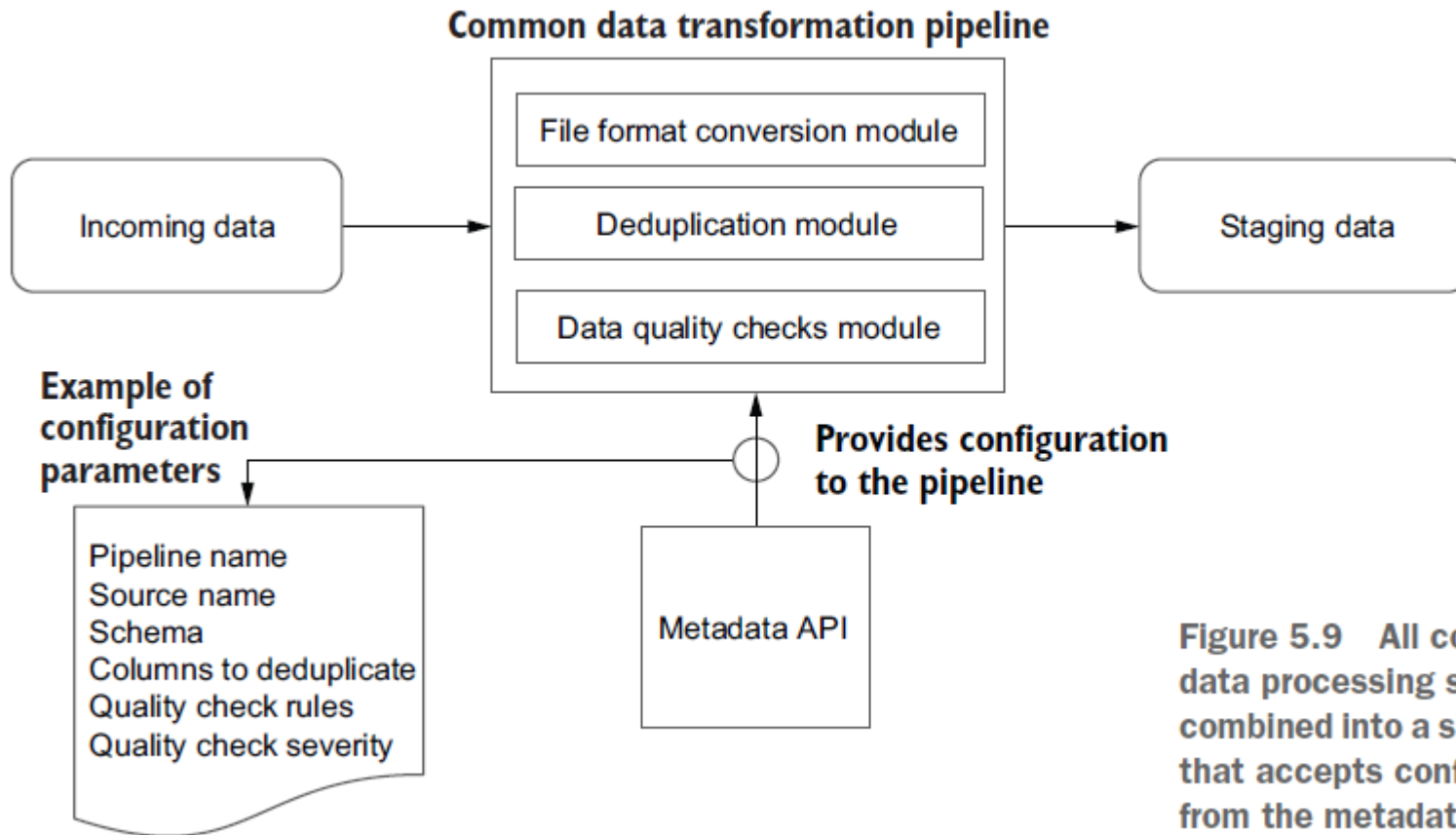
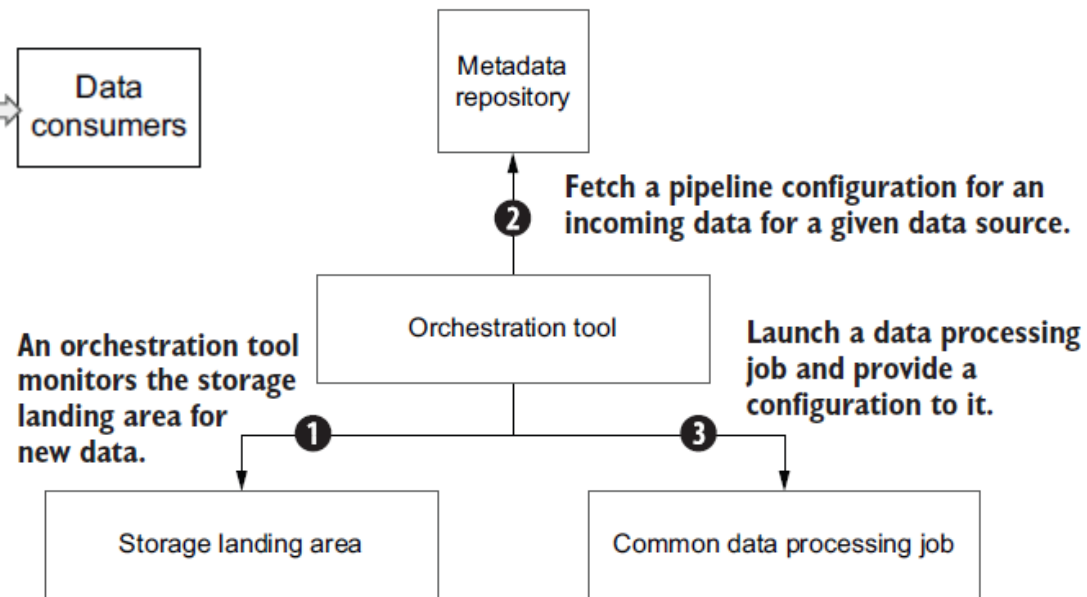
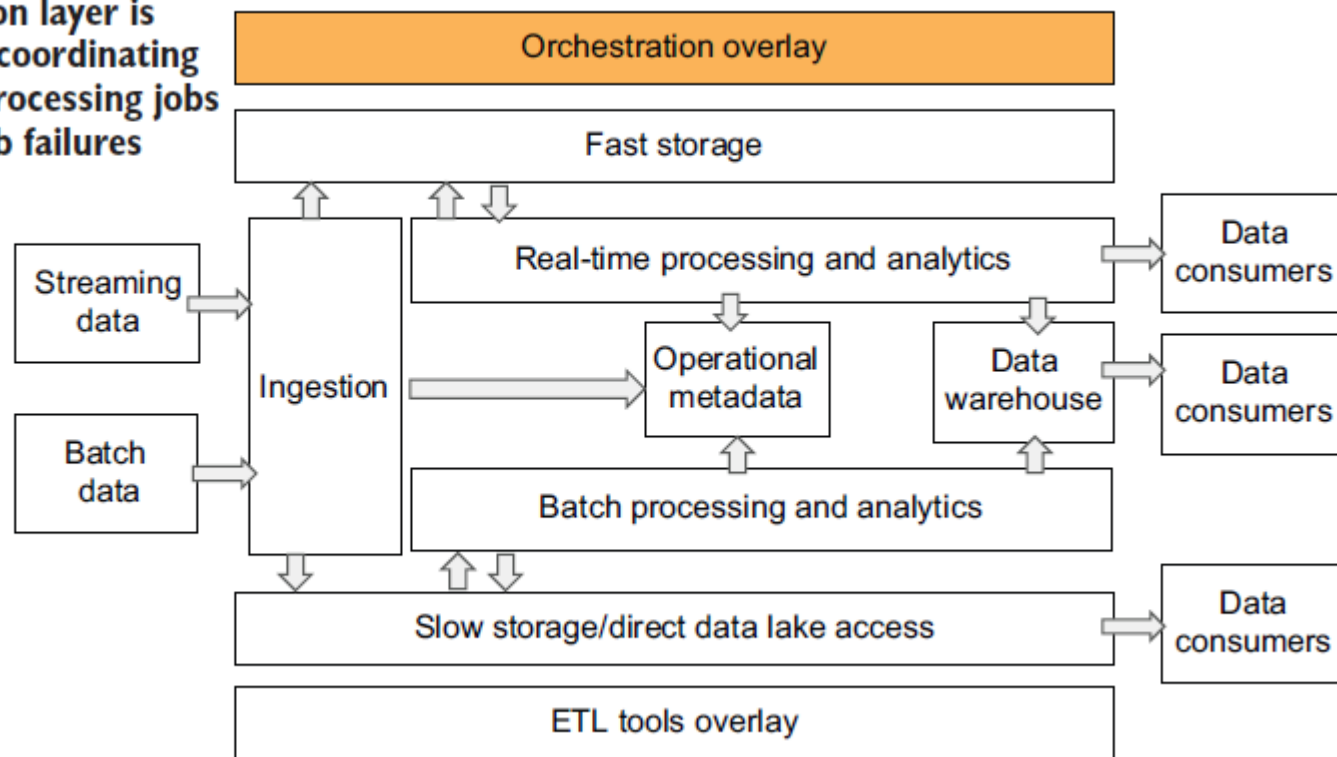


Figure 5.9 All common data processing steps can be combined into a single pipeline that accepts configuration from the metadata API.

설정 가능한 파이프라인

The orchestration layer is responsible for coordinating multiple data-processing jobs and handling job failures and retries.



요약

- 처리 계층은 데이터 플랫폼 구현의 핵심입니다. 모든 데이터 유효성 검사 및 데이터 변환이 수행되고 비즈니스 로직이 적용된다.
- 처리(Processing)은 데이터 웨어하우스에서 수행할 수 있지만, 시스템이 기업에서 매우 중요한 역할을 담당하고 있거나 이를 확장하고 장기적으로 사용할 수 있도록 하려면 플랫폼의 레이크에서 처리를 수행하면 더 나은 결과를 얻을 수 있다.
- 처리 단계에서 데이터는 여러 단계를 거치며 각 단계에서 데이터 변환 및 유효성 검사 논리가 적용되므로 데이터 소스에서 오는 원시 및 정제되지 않은 데이터에서 잘 정의되고 정의된 데이터로 변환될 때 데이터의 "유용성"이 증가한다.
- 데이터 처리 작업은 일반적으로 공통 데이터 처리 단계와 비즈니스 로직 관련 단계의 두 가지 범주로 나눌 수 있다.
- 클라우드 스토리지에서 데이터를 구성하는 방법에 대한 일관되고 명확한 원칙을 갖는 것은 데이터를 읽고 데이터를 쓸 위치와 관련하여 동일한 설계를 따르는 파이프라인을 구축할 수 있게 해주기 때문에 매우 중요하다.

Thank you