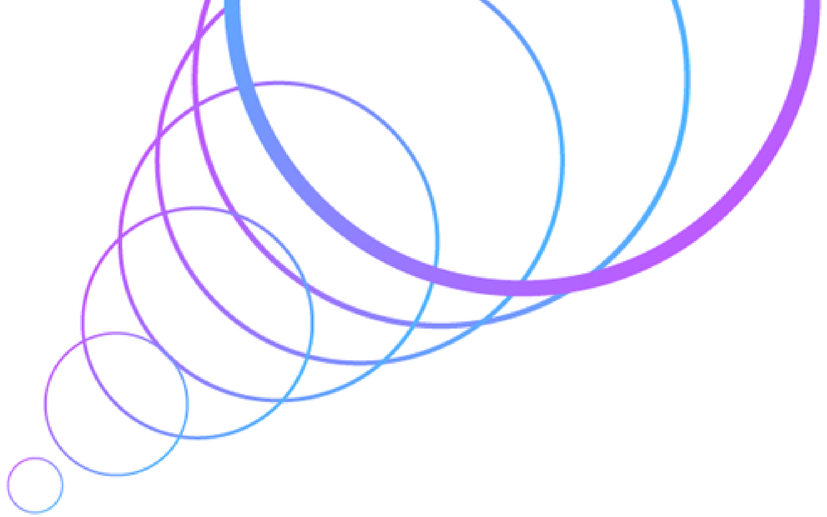


DEVIEW 2021

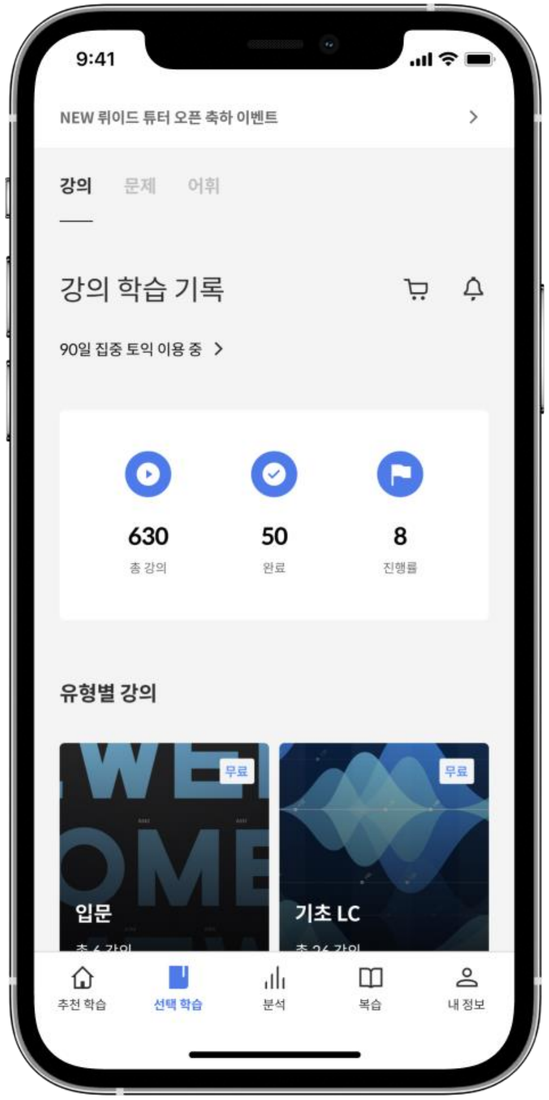
Riiid Tutor의 데이터를 흐르게 하는 기술

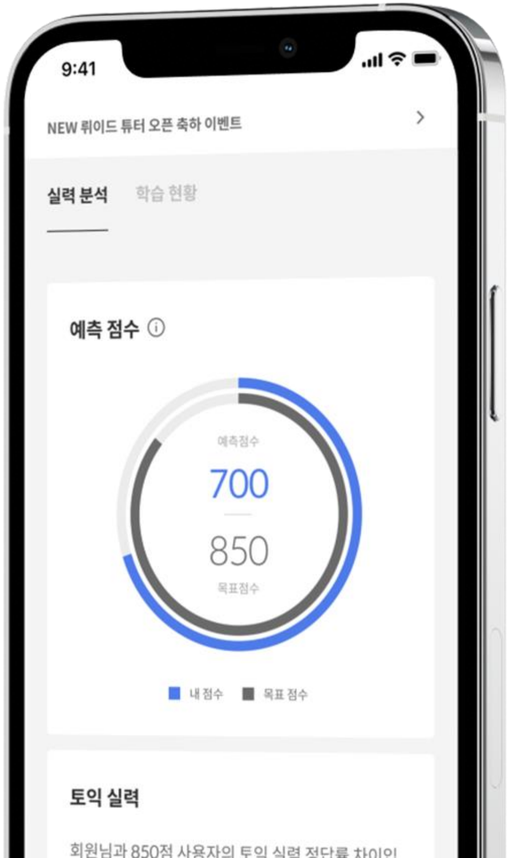
한성민, 최정우 Riiid

CONTENTS

1.Riiid Tutor 서비스 소개

뤼이드 튜터
Riiid Tutor

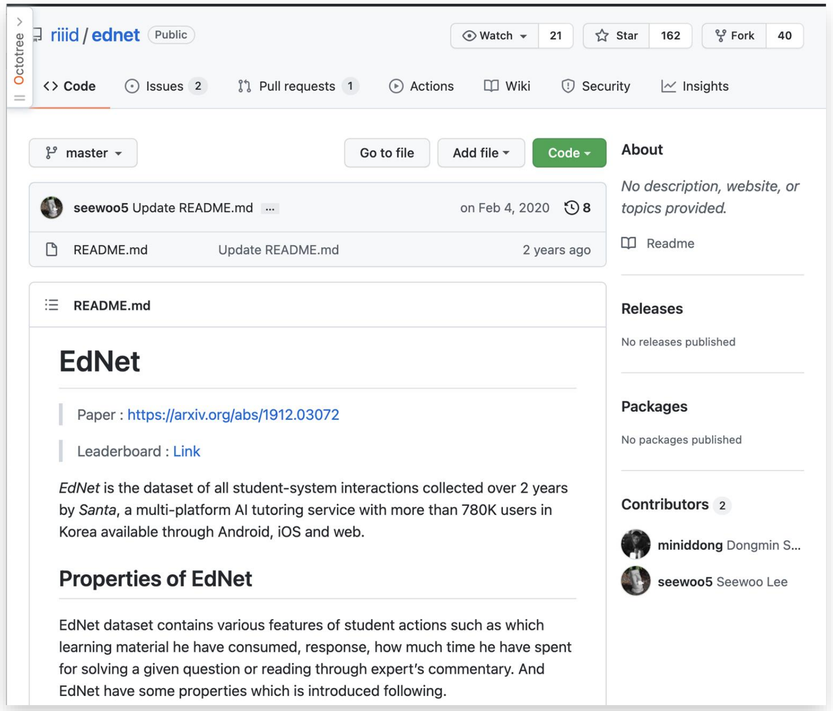




DEVIEW 2021

EdNet

GitHub



https://github.com/riiid/ednet

Arxiv

EdNet: A Large-Scale Hierarchical Dataset in Education

Youngduck Choi[1,2], Youngnam Lee[1], Dongmin Shin[1], Junghyun Cho[1], Seoyon Park[1], Seewoo Lee[1,3], Jineon Baek[1,4], Chan Bae[1,3], Byungsoo Kim[1], and Jaewe Heo[1]

[1] Riiid! AI Research
[2] Yale University
[3] UC Berkeley
[4] University of Michigan
{youngduck.choi,yn.lee,dm.shin,jh.cho,seoyon.park,seewoo.lee, jineon.baek,chan.bae,byungsoo.kim,jwheo}@riiid.co

https://arxiv.org/abs/1912.03072

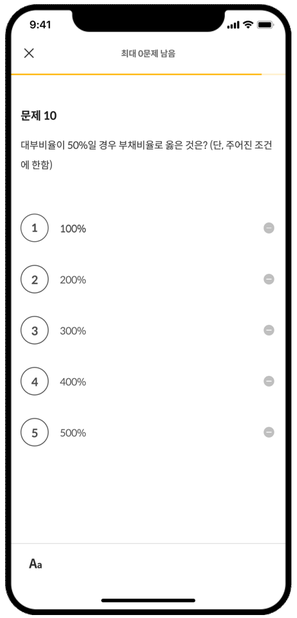
수 많은 도메인과 데이터 관리 비용

DEVIEW 2021

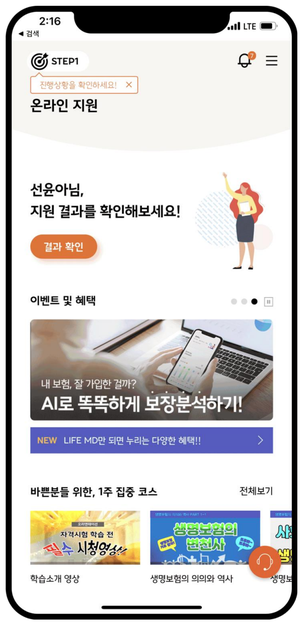
R.INSIDE

영어 시험

공인중개사

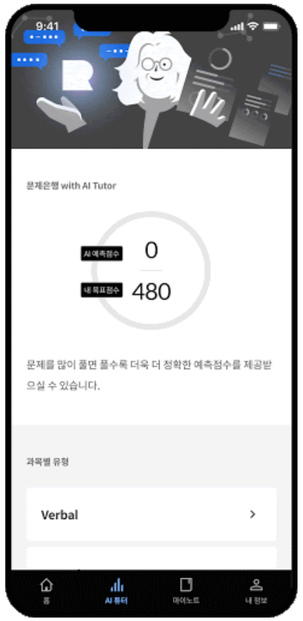


보험설계사

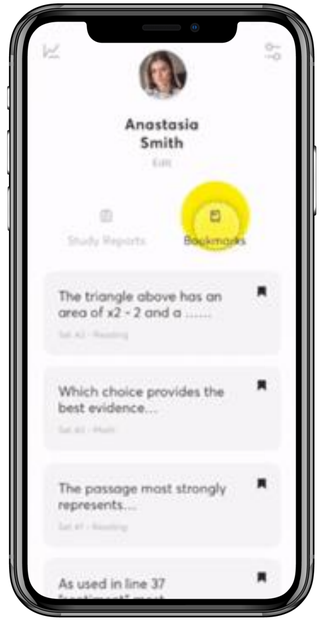


ACT

GMAT

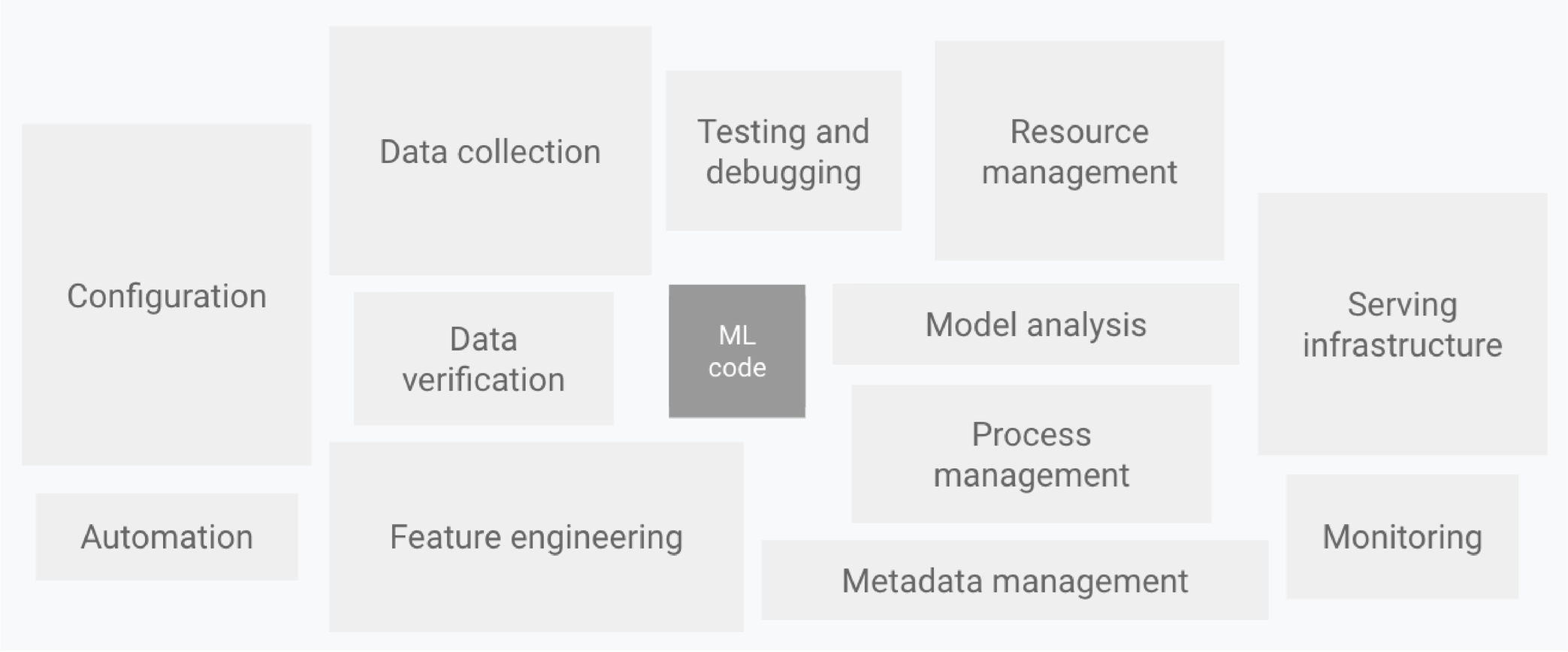


SAT



MLOps Landscape

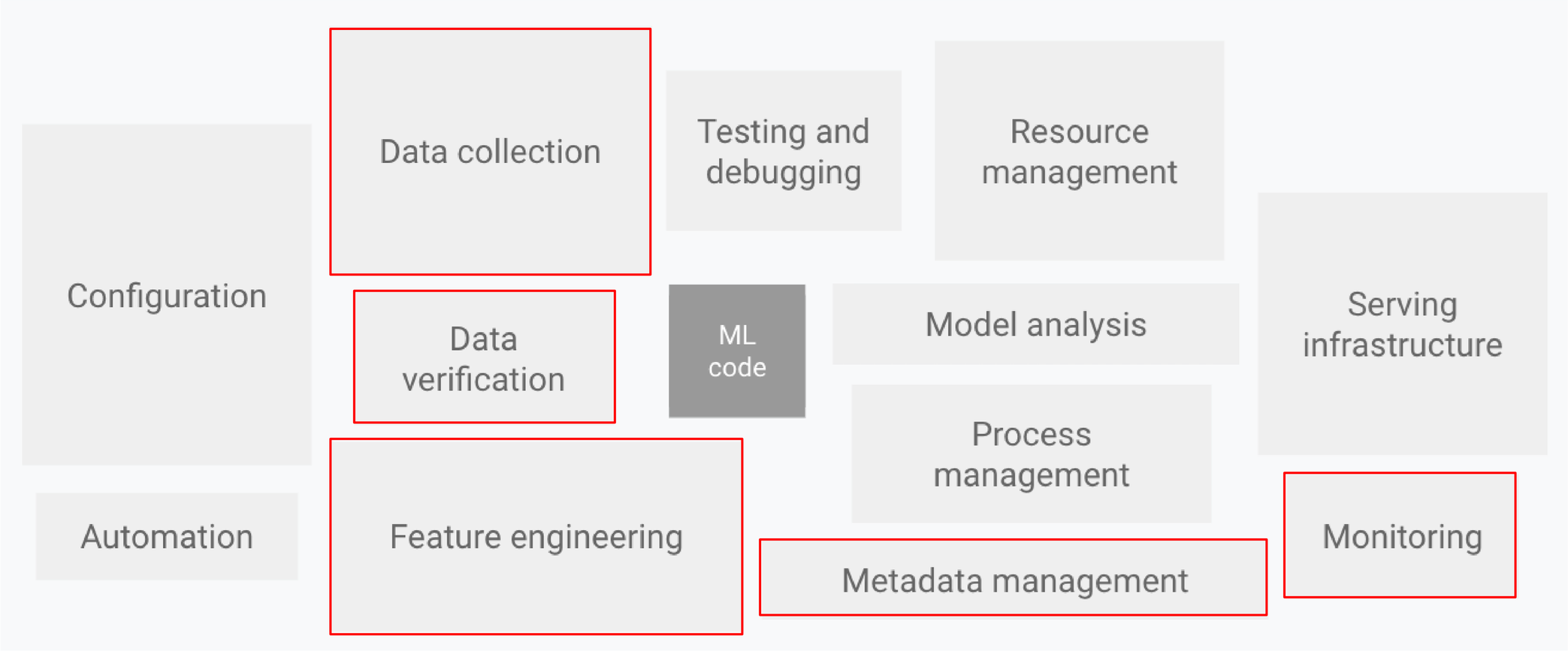
Google Cloud 블로그에 게시된 MLOps 정의



MLOps: Continuous delivery and automation pipelines in machine learning from Google Cloud Blog
https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

MLOps Landscape

이 세션에서는 데이터 분야에 초점

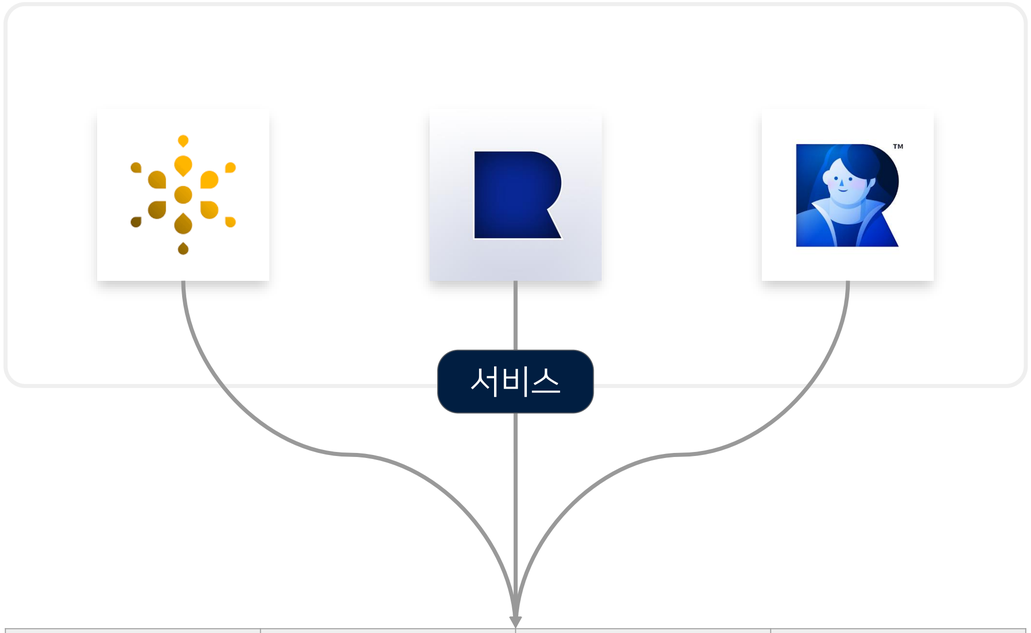


MLOps: Continuous delivery and automation pipelines in machine learning from Google Cloud Blog
https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

2. Riiid Tutor의 데이터와 파이프라인 구조

Riiid Tutor 데이터의 종류

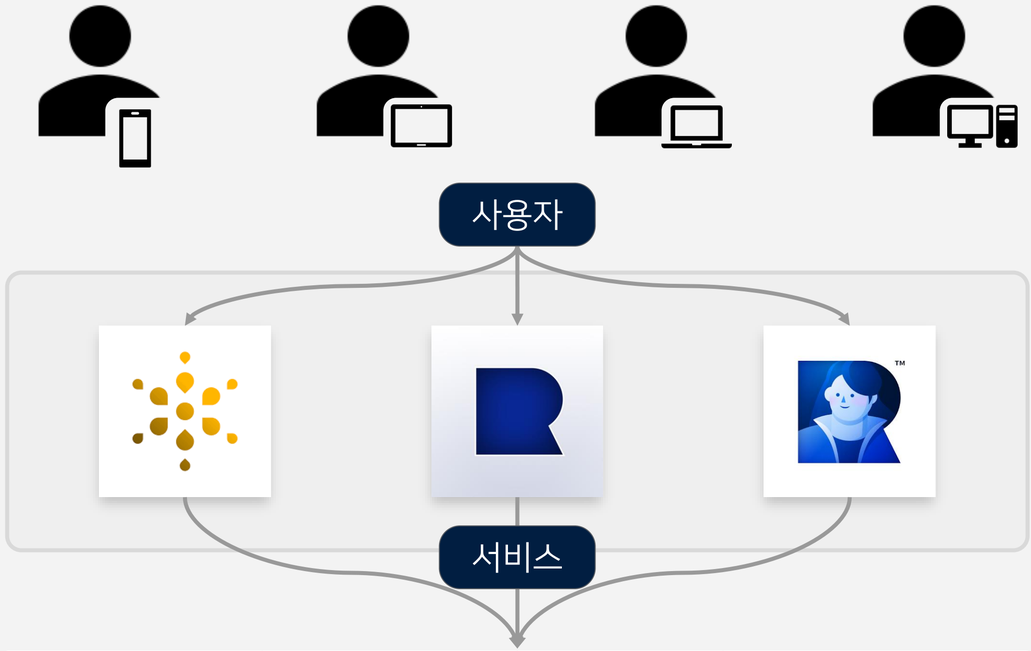
1.학습 문제



QuestionID	Difficulty	Chapter	Answer
Q1	hard	part5	2
Q2	medium	part7	4

데이터

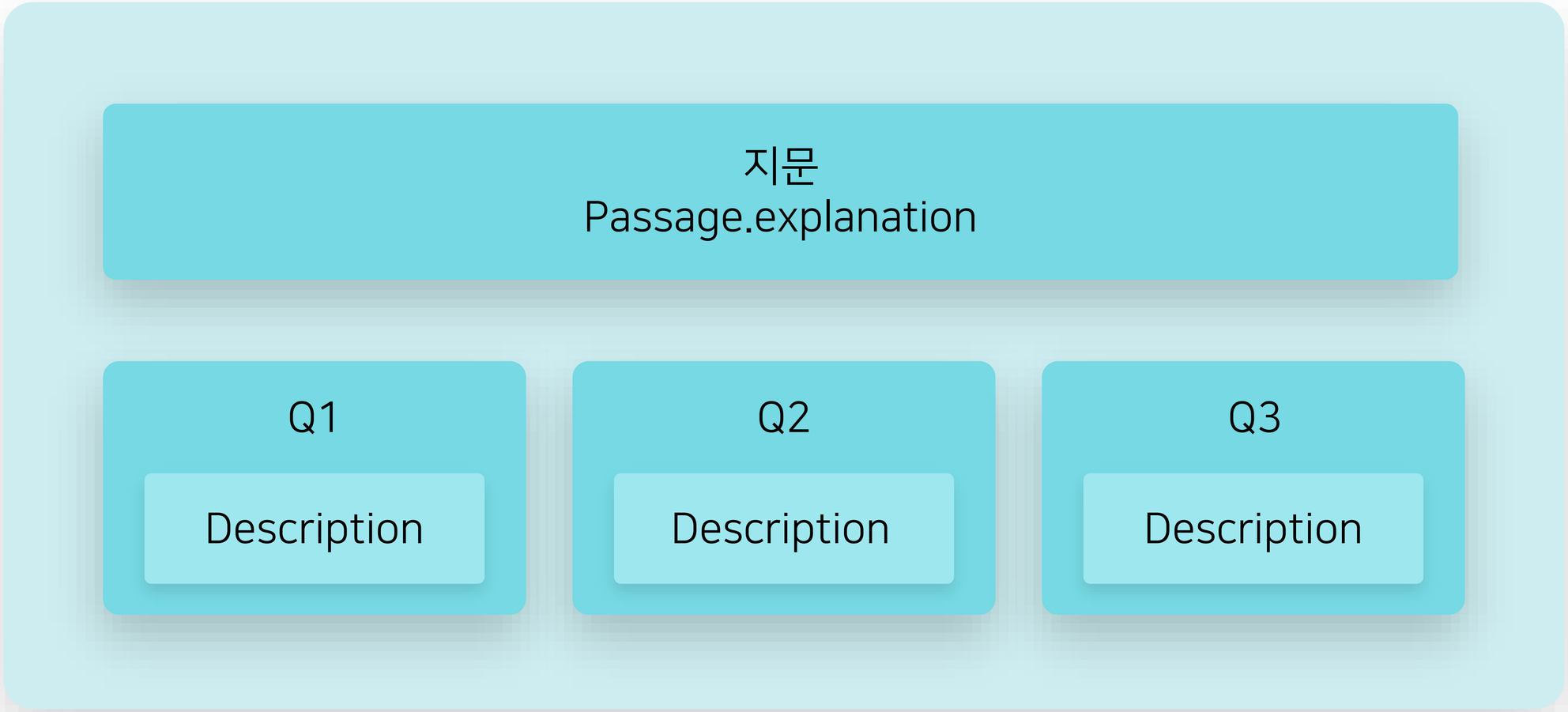
2. 문제 풀이 이력



UserID	QuestionID	Choice	CompletedAt
U1	Q1	3	16:34.12
U1	Q2	4	16:35.20

학습 이력

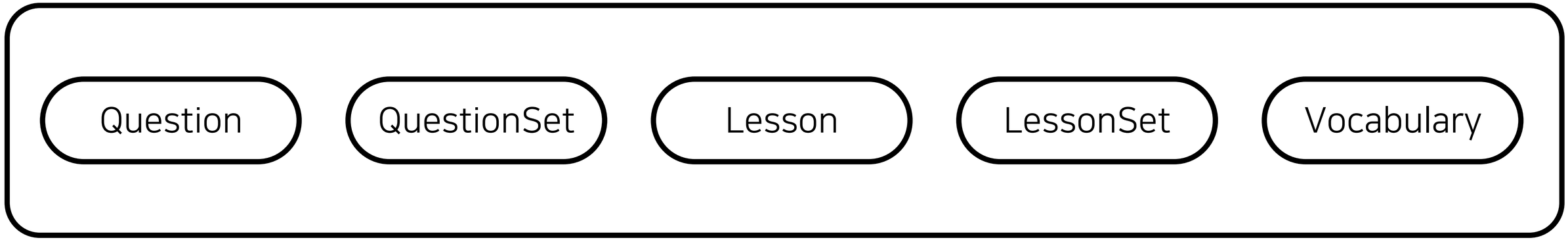
학습 문제 데이터 - 기본 구조



Content

학습 문제 데이터 - 유형

문제에는 다양한 형태가 존재한다!
형태에 따라서 데이터 스키마도 별도로 취급한다. (데이터 엔지니어링 코스트 ↑)



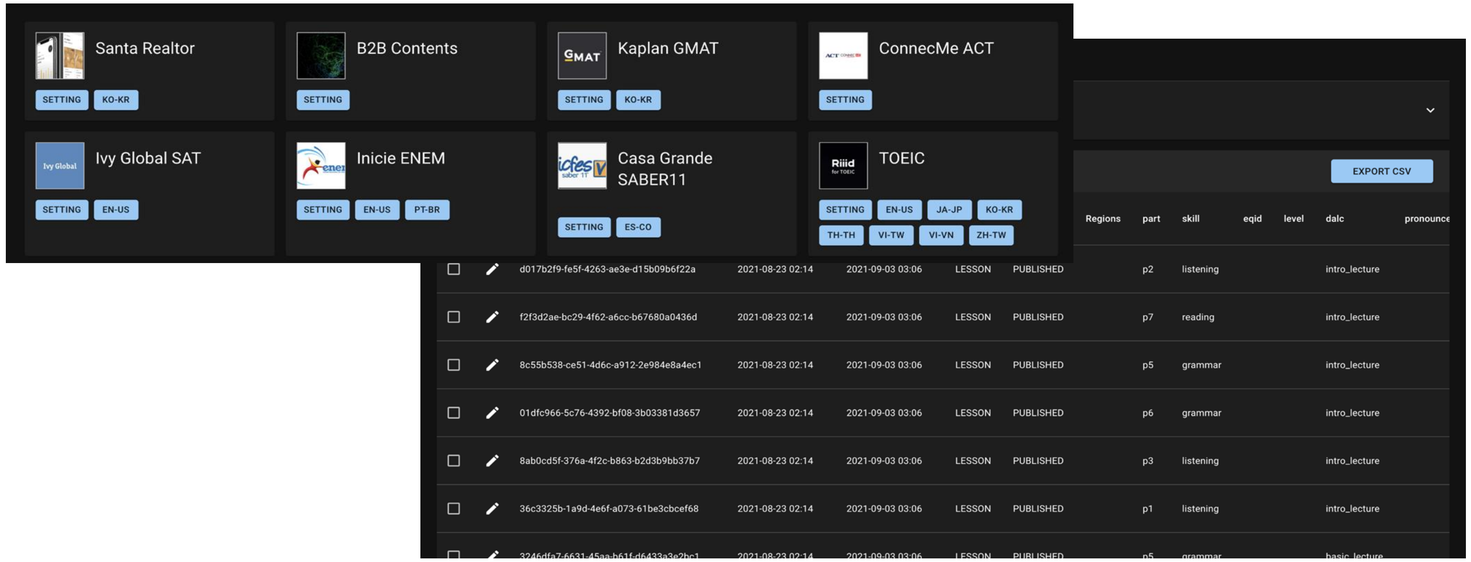
콘텐트 (Content)

학습 문제 데이터 - 관리

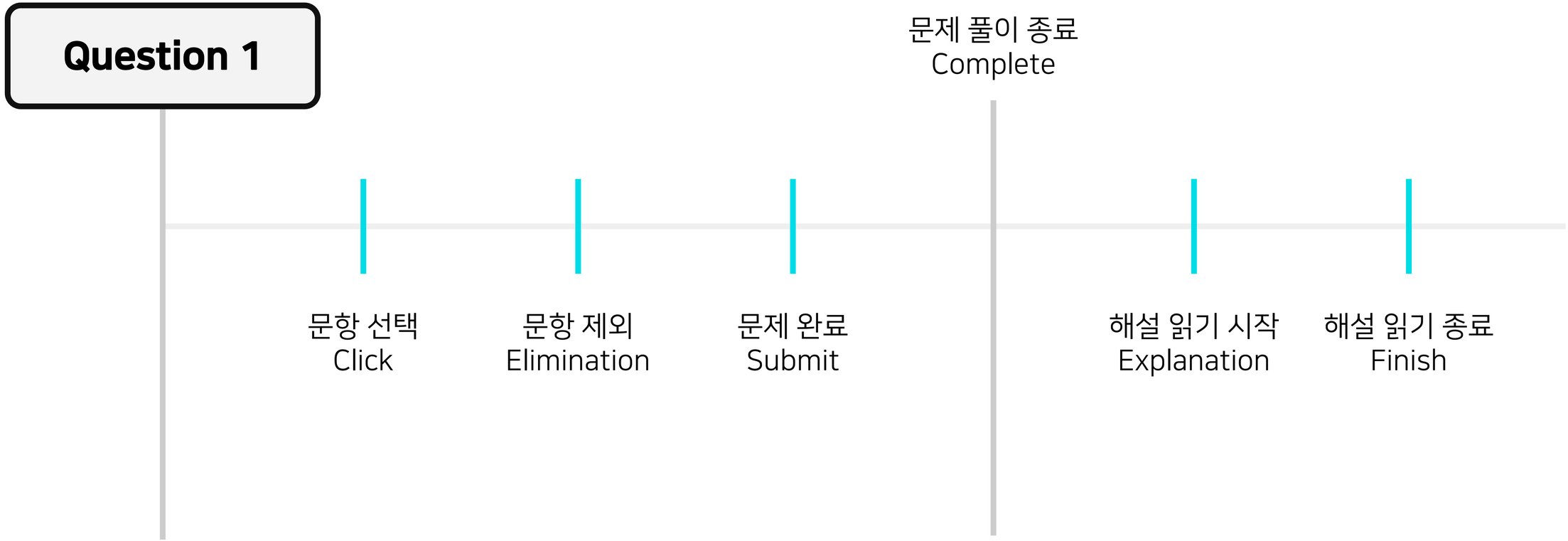
DEVIEW 2021

CMS (Content Management System)

- 내부 콘텐츠 팀에서 문제 등록 및 수정.
- 문제는 한 주에도 여러번 수정이 발생함.
- 변경된 문제 정보를 AI 서빙 시스템에서 어떻게 대응할 것인지 고민하기 시작.

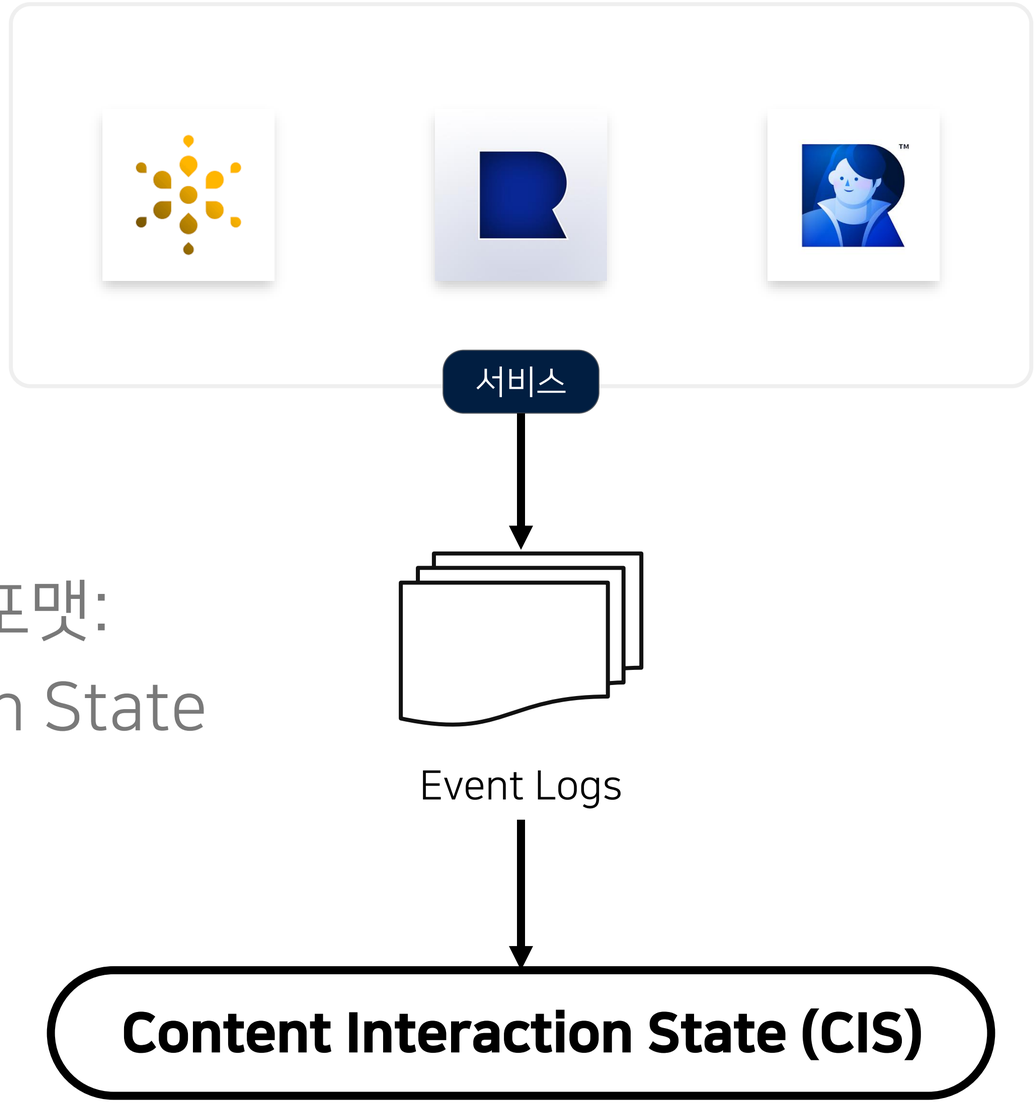


문제 풀이 이력 데이터 - 로깅 구조



사용자가 문제를 푸는 과정에서 다양한 이벤트 로그가 발생하게 된다.

문제 풀이 이력 데이터 - 로깅 포맷



이벤트 로그의 표준 포맷:
Content Interaction State

문제 풀이 이력 데이터 - CIS

```
message ContentInteractionState {
  int64 id = 1;
  string user_id = 2;
  uint64 learning_session_id = 3;
  ContentIdentifier content_identifier = 4;

  oneof state {
    QuestionState question_state = 9;
    ...
  }

  google.protobuf.Timestamp started_at = 12;
  google.protobuf.Timestamp completed_at = 13;
}
```

```
message ContentIdentifier {
  string content_id = 1;
  int64 version_id = 2;
  LanguageTag language_tag = 3;
  Domain domain = 4;
}

message QuestionState {
  int64 id = 8;
  string question_id = 1;
  bool is_completed = 2;
  int64 elapsed_time_ms = 3;
  google.protobuf.BoolValue is_correct = 4;
  message Objective {
    repeated uint32 user_answer = 1;
    repeated uint32 eliminated_choice_indices = 2;
  }
  ...
}
```

protobuf3 스키마로 인해
Complex struct 구조의 로그

데이터 포맷

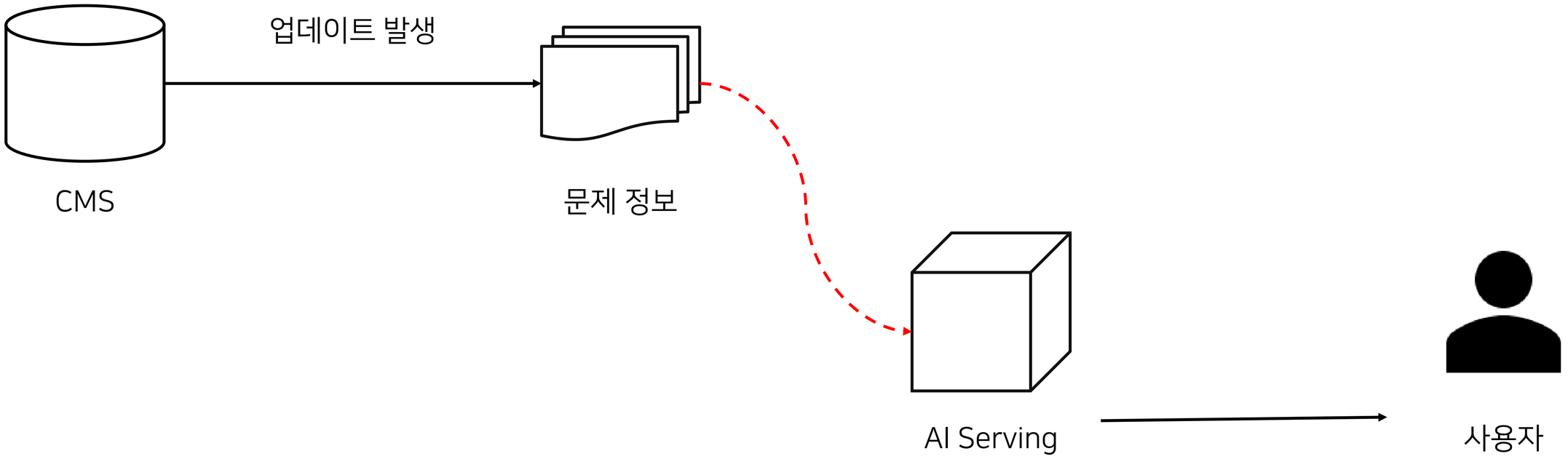
학습 문제 데이터 (Content)

```
{
  "contentId": "bdd8b79a-8137-4e9e-91df-d41e54565a4a",
  "contentType": "QUESTION_SET",
  "contentBody": {
    "questionSet": {
      "questions": [ {
        "description": "...",
        "explanation": "...",
        "objective": { ... }
      }]
    }
  },
  "domain": "TOEIC",
  ...
  "tags": [
    {
      "key": "difficulty",
      "value": { "stringValue": "hard" }
    }
  ],
  "versionId": 233195
}
```

문제 풀이 이력 데이터 (Interactions)

```
CIS {
  id = "c1347a04-92ff-4fbe-b624-1faa36aedf99"
  user_id = "1fc31378-3cf6-4473-b294-9ee72420b5e8"
  learning_session_id = "9d3847a9-93e6-4f67-87e8-68f4bb0f6217"
  content_identifier = ContentIdentifier {
    content_id = "bdd8b79a-8137-4e9e-91df-d41e54565a4a"
    version_id = 233195
    language_Tag = LanguageTag { language=KO, country=KR }
    domain = TOEIC
  }
  question_state = QuestionState {
    id = "caaaf89f-5021-4f3a-b9a9-78780ced0723"
    question_id = "question-1"
    started_at = 1633689817
    ended_at = 1633691252
    ...
  }
}
```

데이터 파이프라인 - Serving system



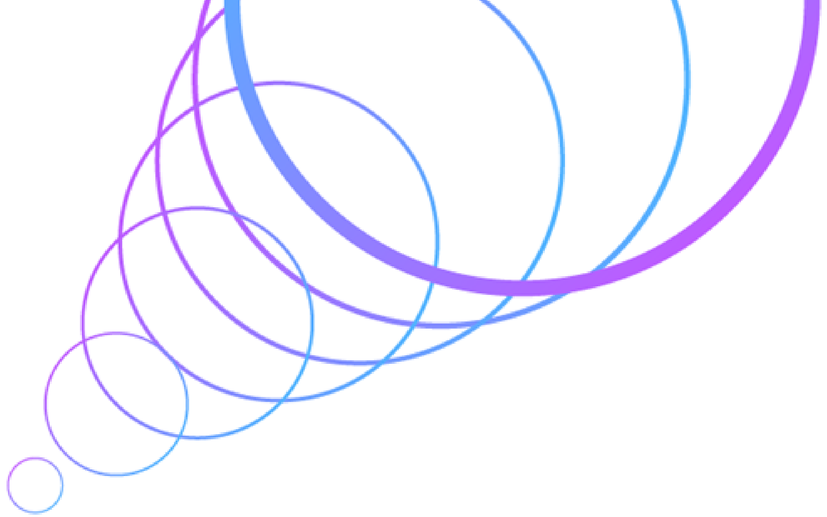
문제 정보가 업데이트 될 때 어떻게 서빙에 반영할까?

데이터 파이프라인 요구사항

- 데이터 스키마의 변화가 생기면 Side-effect를 어떻게 쉽게 발견할 수 있을까?
- 데이터가 특정 주기마다 업데이트 될 때,

  업데이트 된 정보를 서빙 시스템에 즉각적으로 반영할 순 없을까?
- 비정형으로 수집된 로그를 막대한 전처리 과정 없이

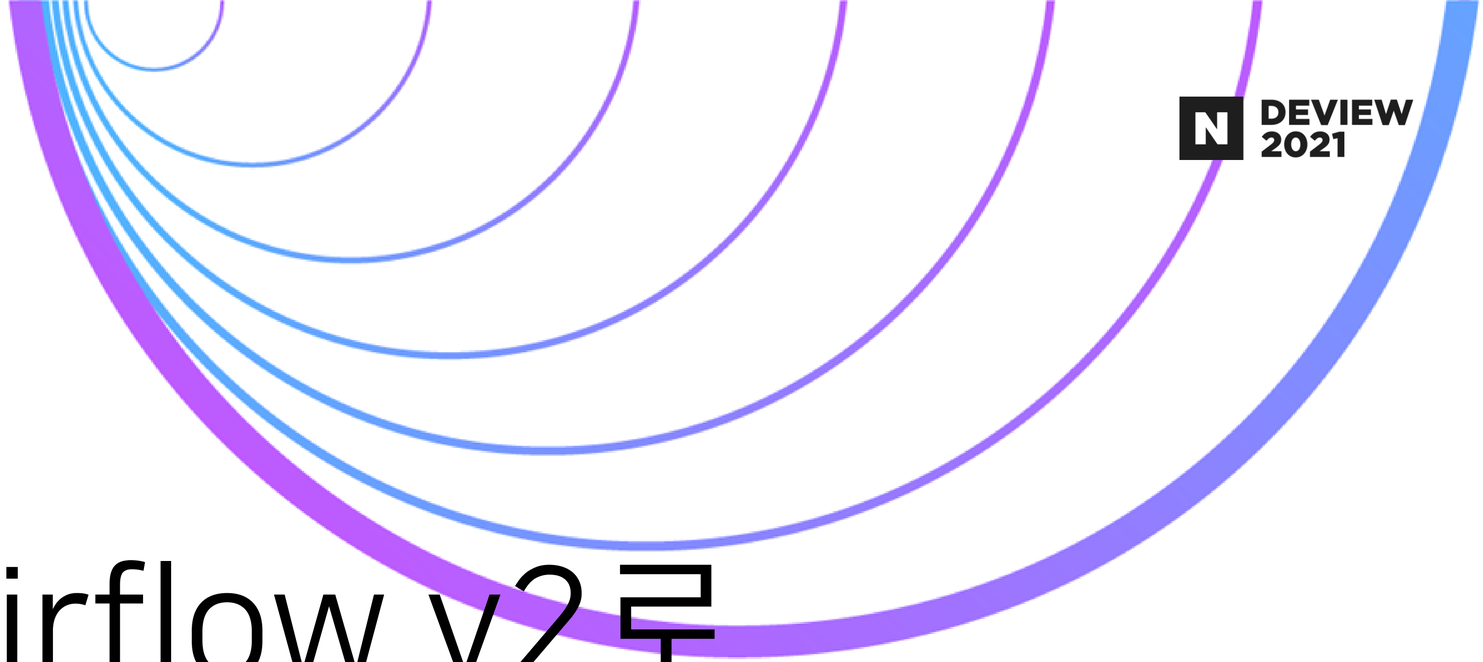
  쉽게 보관할 수 있는 방법이 없을까?



DEVIEW 2021

Riiid Tutor의 데이터를 흐르게 하는 기술

한성민, 최정우 Riiid

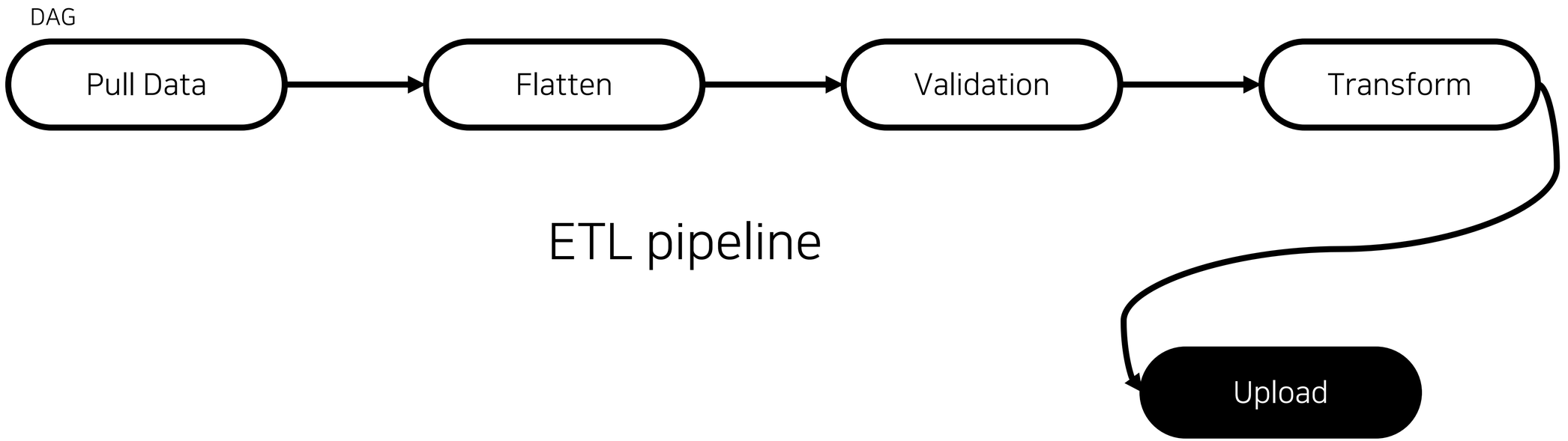


DEVIEW 2021

3. Airflow v2로 데이터 워크플로우 설계하기

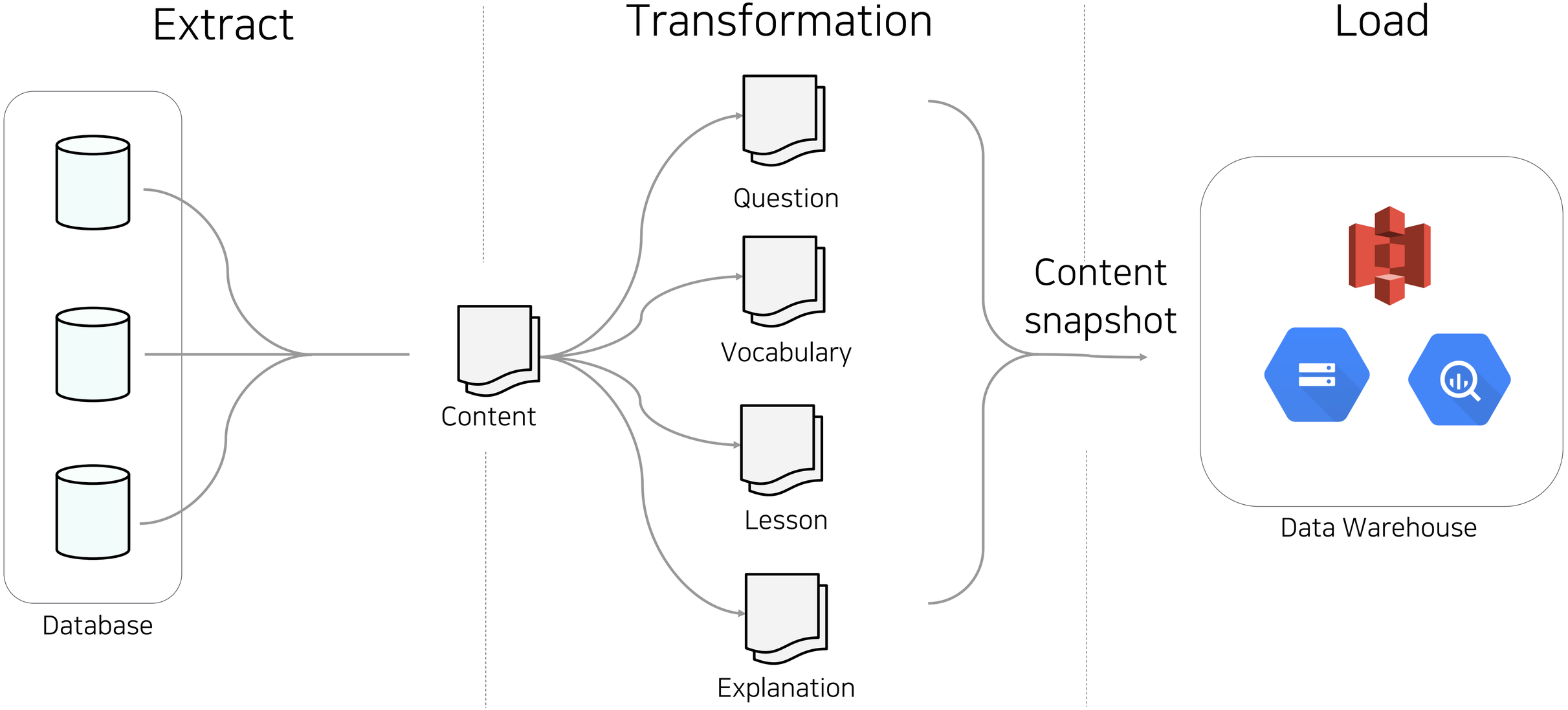
Airflow 소개

: Airflow is an open-source workflow management platform



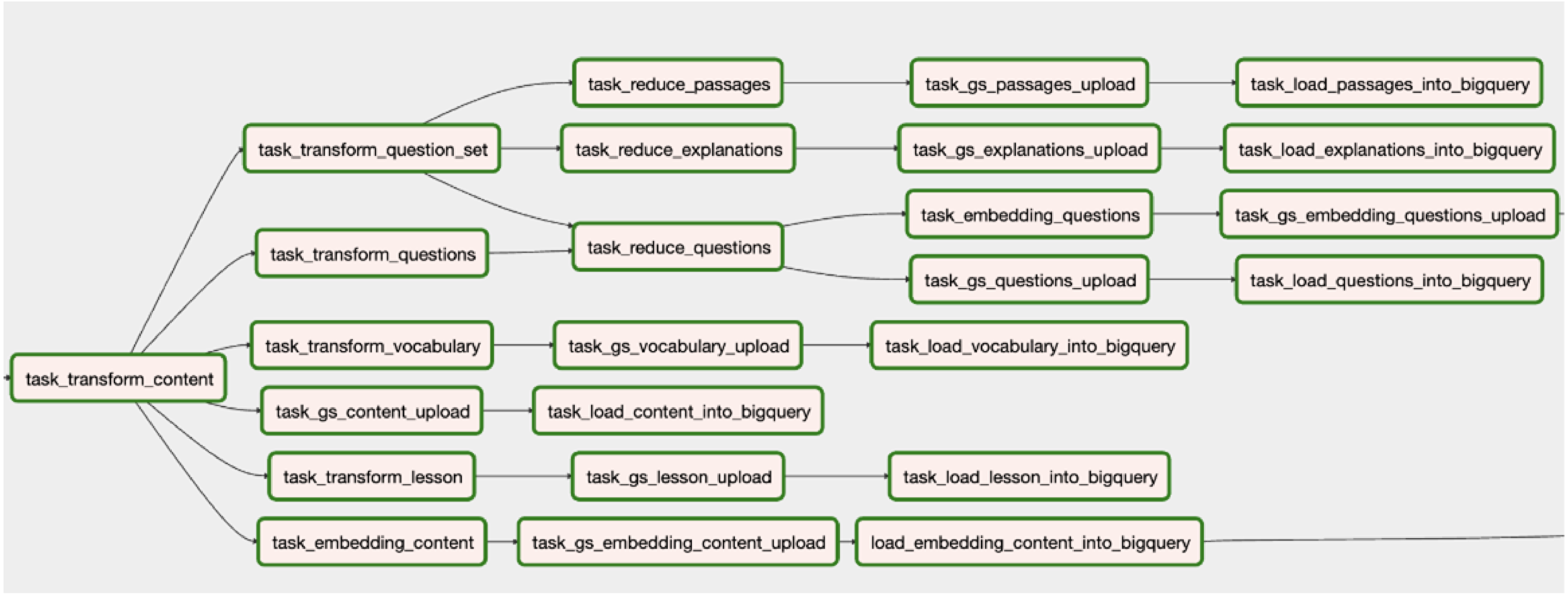
ETL pipeline

ETL on Riiid Tutor Data



ETL DAG 예시

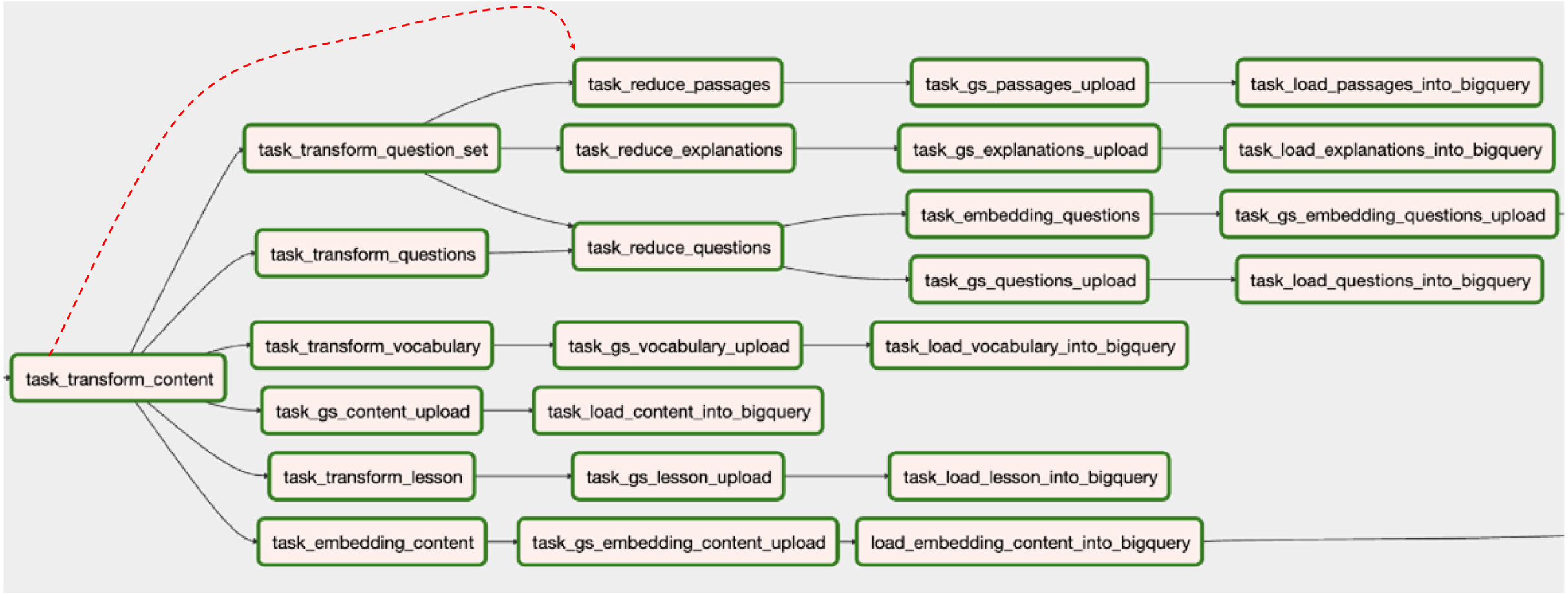
가장 상위레벨의 문제 정보 데이터인 Content 데이터로부터 하위레벨 데이터로 변형시키고 load하는 DAG



기존 ETL DAG의 문제점

'작업' 단위가 실행되는 순서(경로)와, 실제 '데이터'가 흐르는 경로가 서로 다를 수 있음

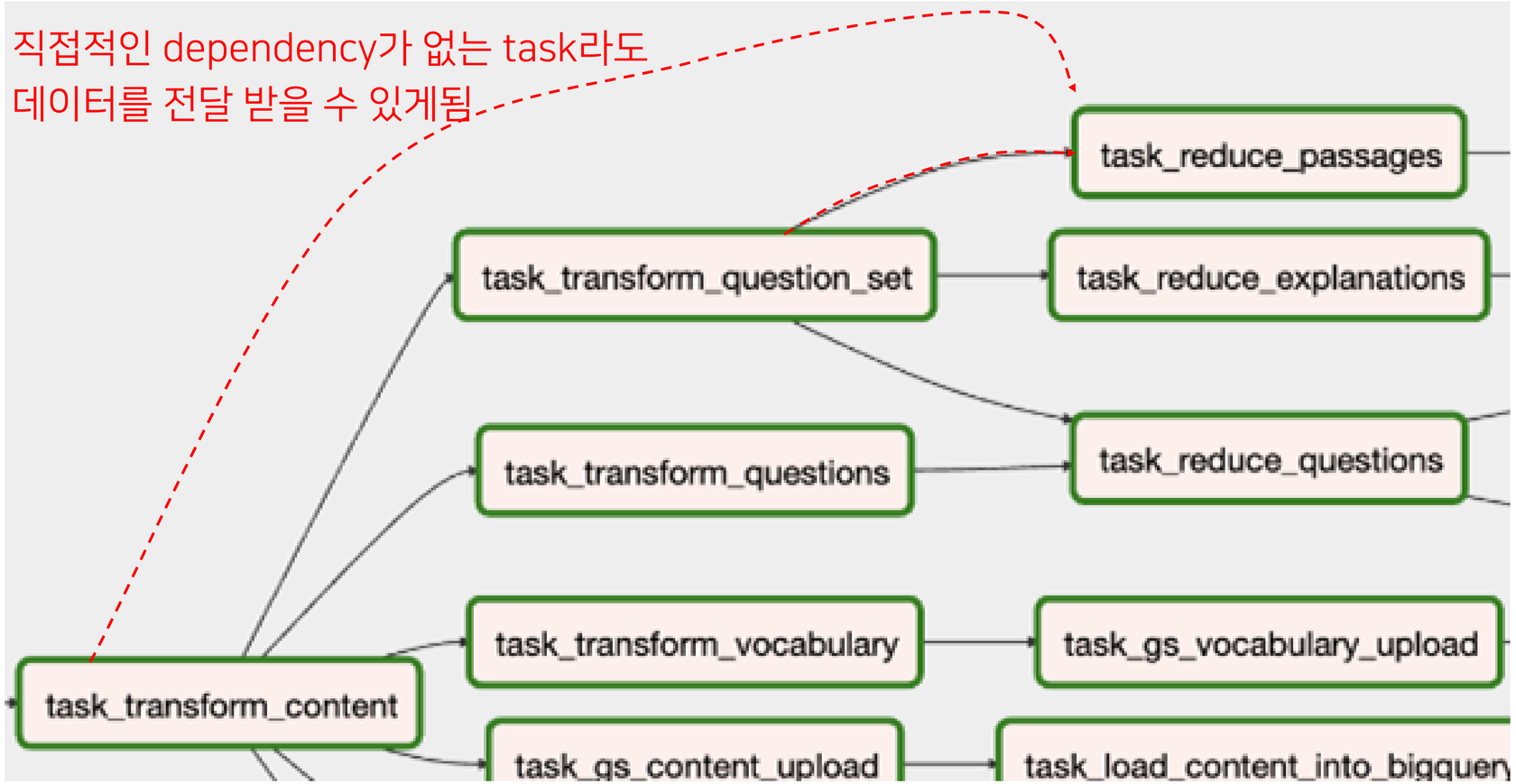
데이터 이동 경로
task 실행 순서



기존 ETL DAG의 문제점

'작업' 단위가 실행되는 순서(경로)와, 실제 '데이터'가 흐르는 경로가 서로 다를 수 있음

데이터 이동 경로
task 실행 순서



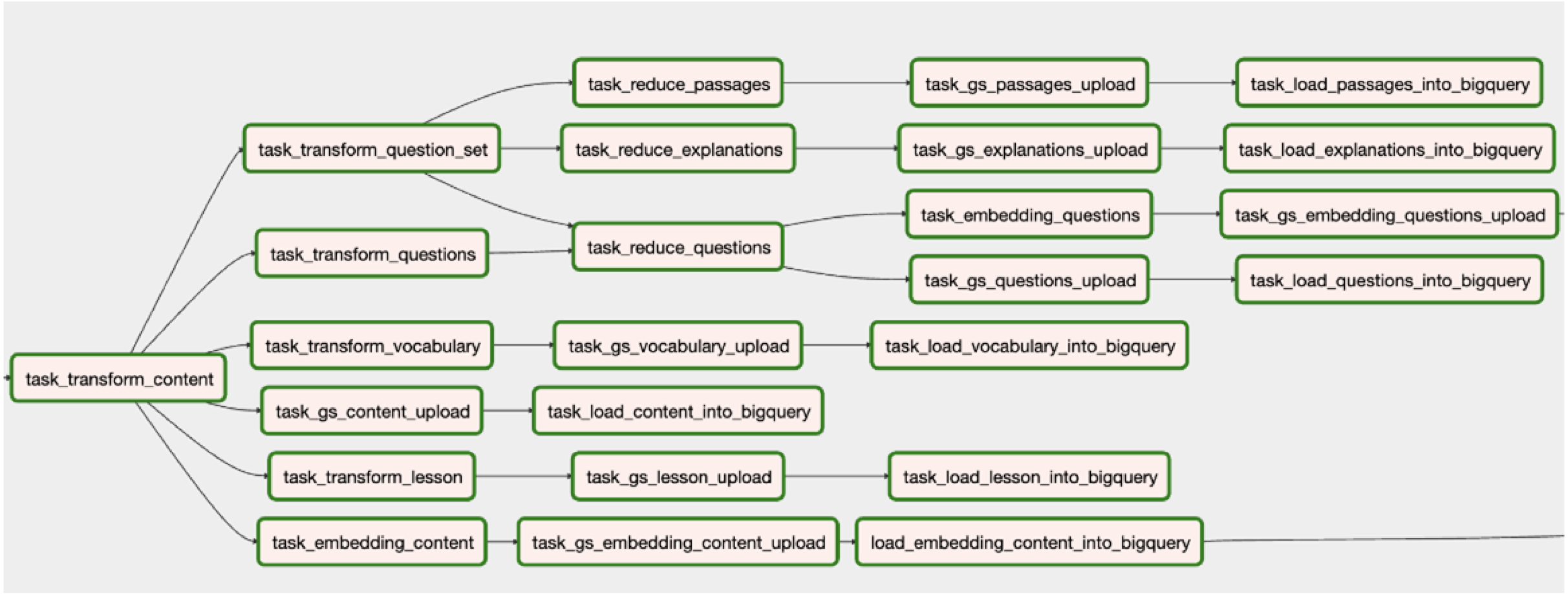
기존 ETL DAG의 문제점

DEVIEW 2021

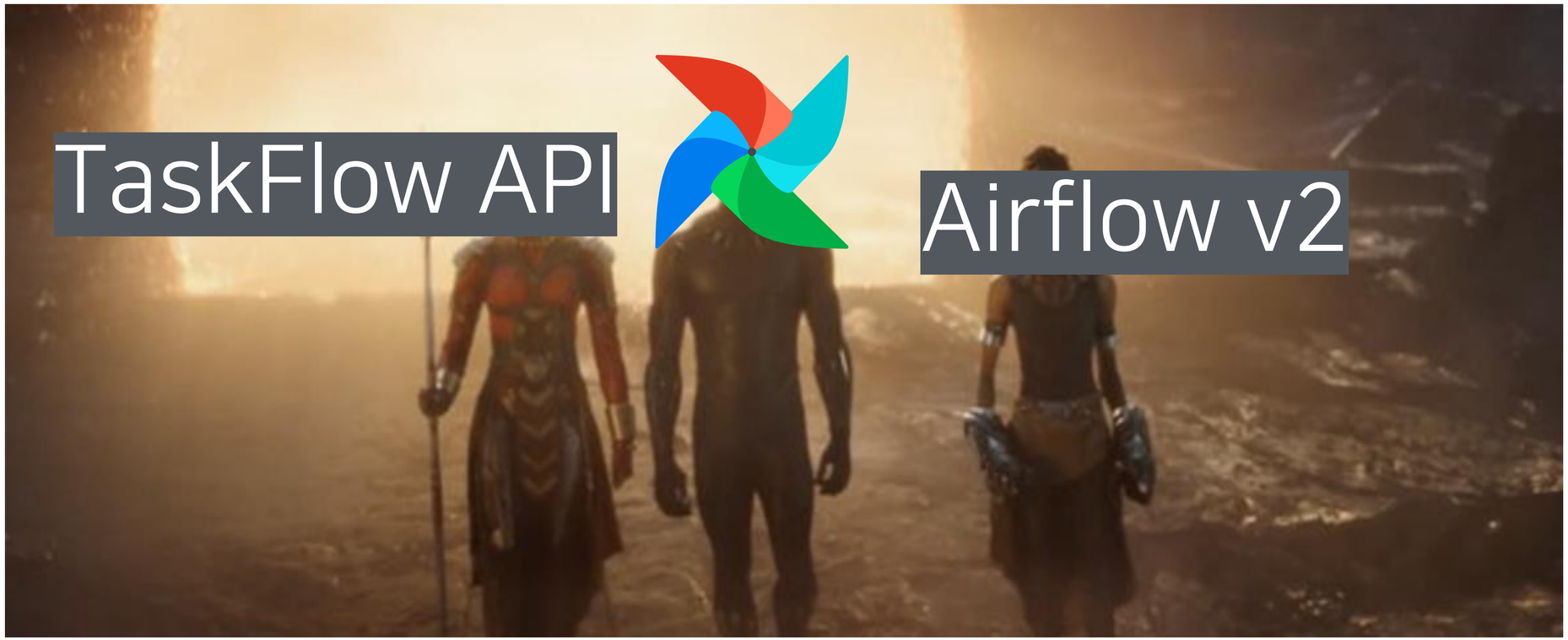
데이터 이동 경로

task 실행 순서

DAG구조와 Task 파악의 어려움



Airflow v2



TaskFlow API: @task, XComArg

<V1>

```python
def push_function2(**context):
    context['task_instance'].xcom_push(key='pushed_value', value='xcom_push')

def pull_function(**context):
    value = context['ti'].xcom_pull(task_ids='push_info', key="pushed_value")

push_info = PythonOperator(
    task_id='push_info',
    python_callable=push_function,
    dag=dag,
    provide_context=True,
    xcom_push=True,
)

pull_info = PythonOperator(
    task_id='pull_info',
    python_callable=pull_function,
    dag=dag,
    provide_context=True,
    xcom_push=True,
)

push_info >> pull_info
```

함수 내에서 데이터를 push, pull 할 수 있기 때문에, task간에 의존성과는 상관없이 데이터를 전달하고 받을 수 있게 됩니다.

<V2>

```python
@task
def push_function():
    return 'xcom_test'

@task
def pull_function(data):
    value = data

t1 = push_function()   # type(t1) == XcomArg
pull_function(t1)
```

task의 return 값을, 다음 task에
그대로 전달하기 때문에,
task 의존성과 data 의존성이 일치합니다.

TaskFlow API: @task, XComArg

DEVIEW 2021

`Operator` → `Operator`

```
my_task1_result = PythonOperator(
    task_id="task1",
    python_callable=my_task1,
)
my_task2_result = PythonOperator(
    task_id="task2",
    python_callable=my_task2,
    op_kwargs={"a": my_task1_result.output}
)
```

`@task` → `@task`

```
my_task1_result = my_task1()
my_task2_result = my_task2(my_task1_result)
```

`Operator` → `@task`

```
my_task1_result = PythonOperator(
    task_id="task1",
    python_callable=my_task1,
)
my_task2(my_task1_result.output)
```

`@task` → `Operator`

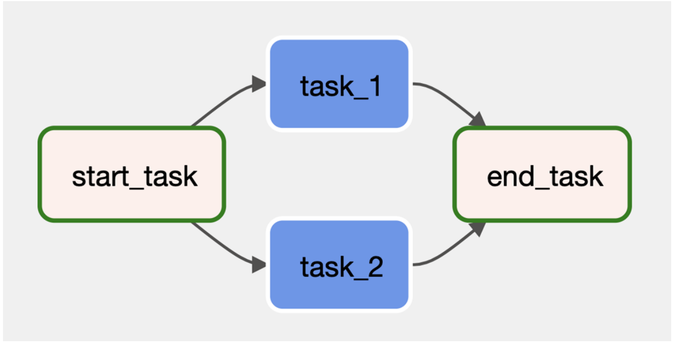
```
my_task1_result = my_task1()
t2 = PythonOperator(
    task_id="task2",
    python_callable=my_task2,
    op_kwargs={"a": my_task1_result}
)
```

TaskFlow API: TaskGroup

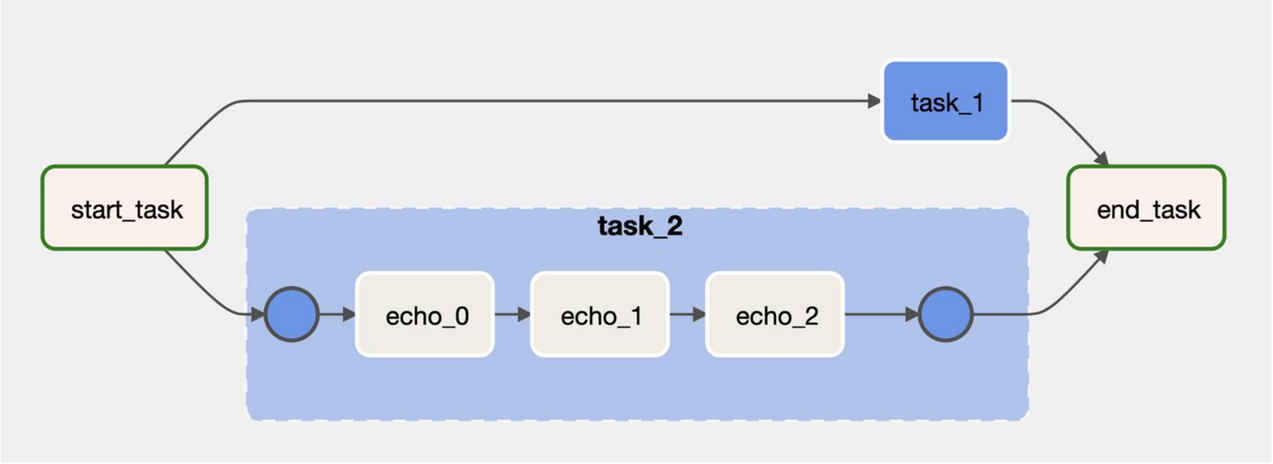
그룹화
```python
with TaskGroup(group_id="task_1") as tg1:
    prev_task = None
    for i in range(3):
        current_task = BashOperator(
            task_id="echo_{}".format(i),
            bash_command="echo 1"
        )
        if prev_task:
            prev_task >> current_task
        prev_task = current_task
```

그룹화
```python
with TaskGroup(group_id="task_2") as tg2:
    prev_task = None
    for i in range(3):
        current_task = BashOperator(
            task_id="echo_{}".format(i),
            bash_command="echo 2"
        )
        if prev_task:
            prev_task >> current_task
        prev_task = current_task
```

<접혔을 때 모습>



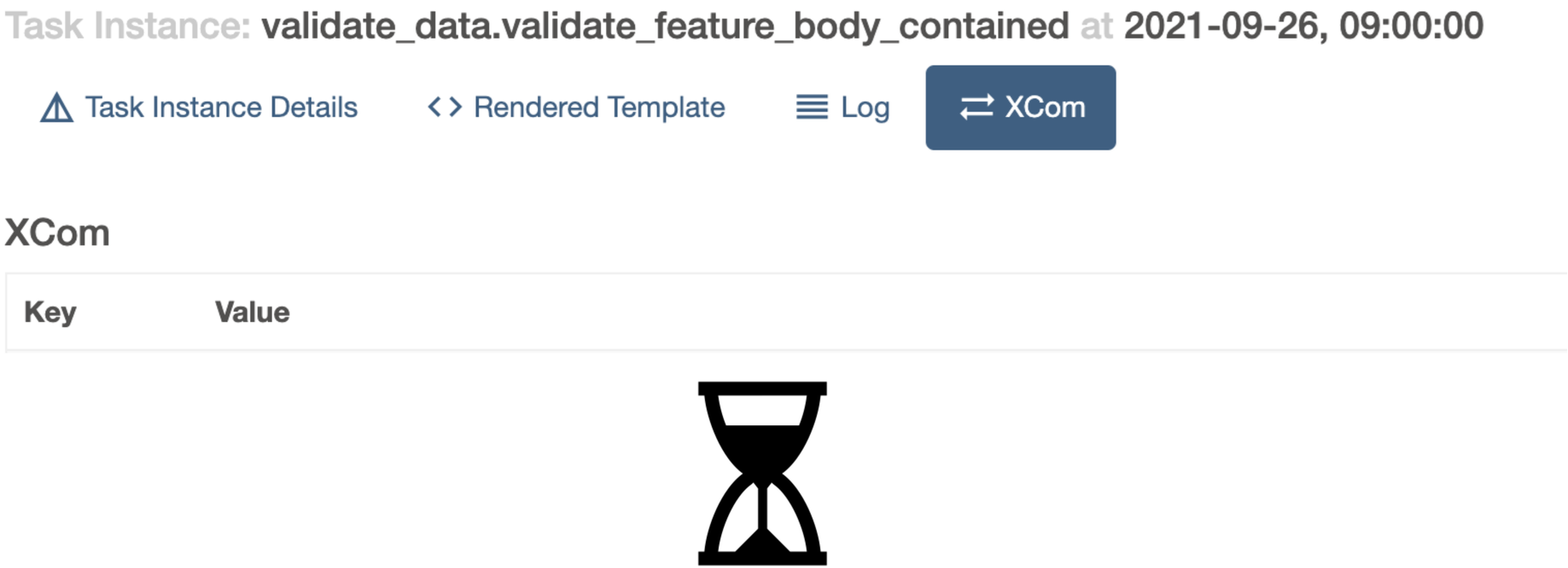
<펼쳐졌을 때 모습>



TaskFlow API: Custom XCom backend

DEVIEW 2021

Task간에 어떤 데이터를 주고 받았는지 확인하기 → Loading이 너무 길어진다면?



Default meta-store의 용량적 한계!

TaskFlow API: Custom XCom backend

DEVIEW 2021

```python
class S3XComBackend(BaseXCom):
    PREFIX = "xcom_s3://"
    BUCKET_NAME = "riiid-airflow-xcom-backend-dev"

    @staticmethod
    def serialize_value(value: Any):
        if isinstance(value, pd.DataFrame):
            hook = S3Hook()
            key = "data_" + str(uuid.uuid4())
            _ = hook.load_bytes(
                bucket_name=S3XComBackend.BUCKET_NAME,
                key=key,
                bytes_data=value.to_csv(index=False).encode(),
                replace=True,
            )
            # Append prefix to persist information that the fi
            value = S3XComBackend.PREFIX + key
        return BaseXCom.serialize_value(value)
```

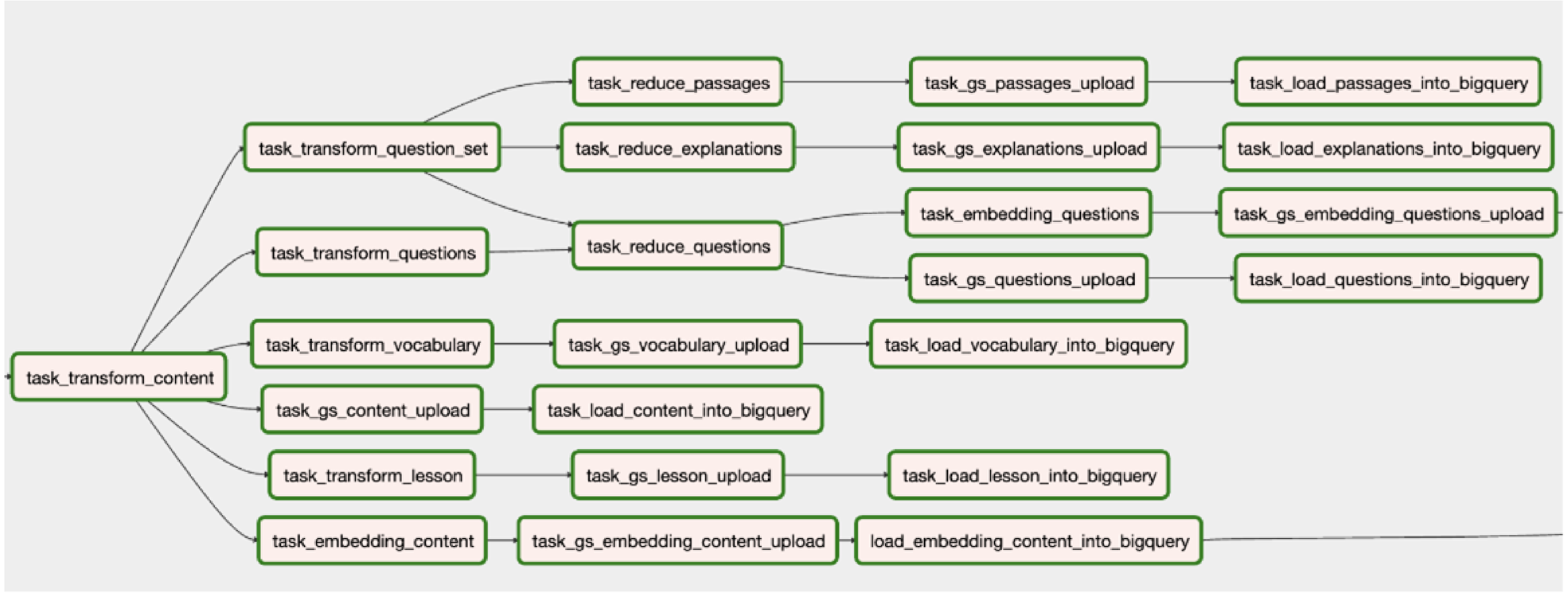
S3 경로를 return

```python
@task
def generate_data() -> pd.DataFrame:
    data = {"a": [1, 2, 3], "b": [12, 13, 14]}
    return pd.DataFrame(data)

@task
def process_data(df: pd.DataFrame):
    df["a"] = df["a"] + 2
    print(df.head())

with DAG(
    "using_gcs_for_xcom",
    default_args={'owner': 'airflow'},
    start_date=days_ago(2),
    schedule_interval=None,
) as dag:
    df = generate_data()
    process_data(df)
```

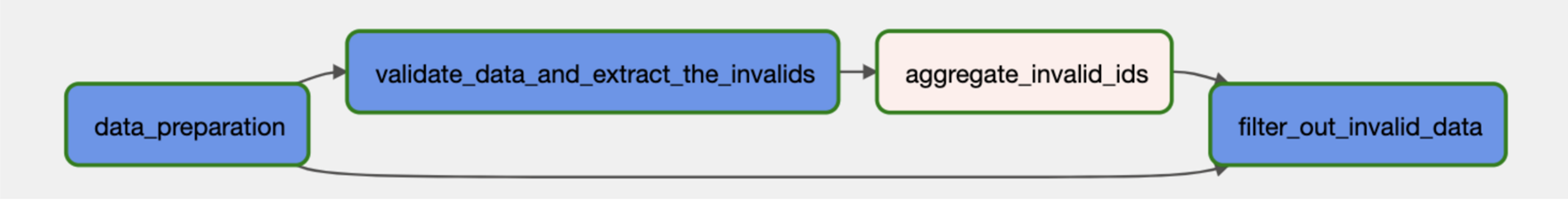
DataFrame으로
바로 주고받을 수 있게 됨

Before & After

<Before>

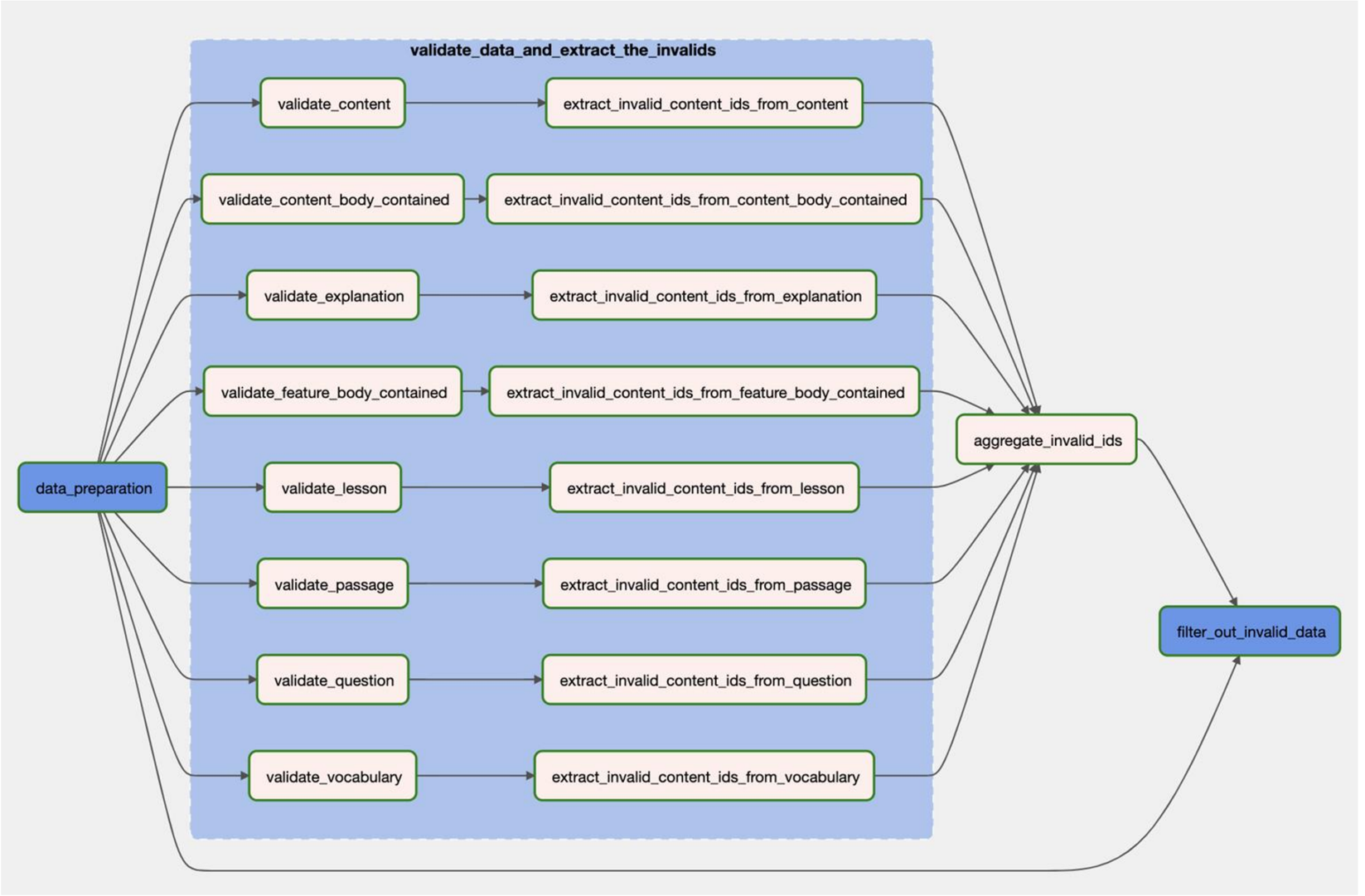


<After>



Before & After

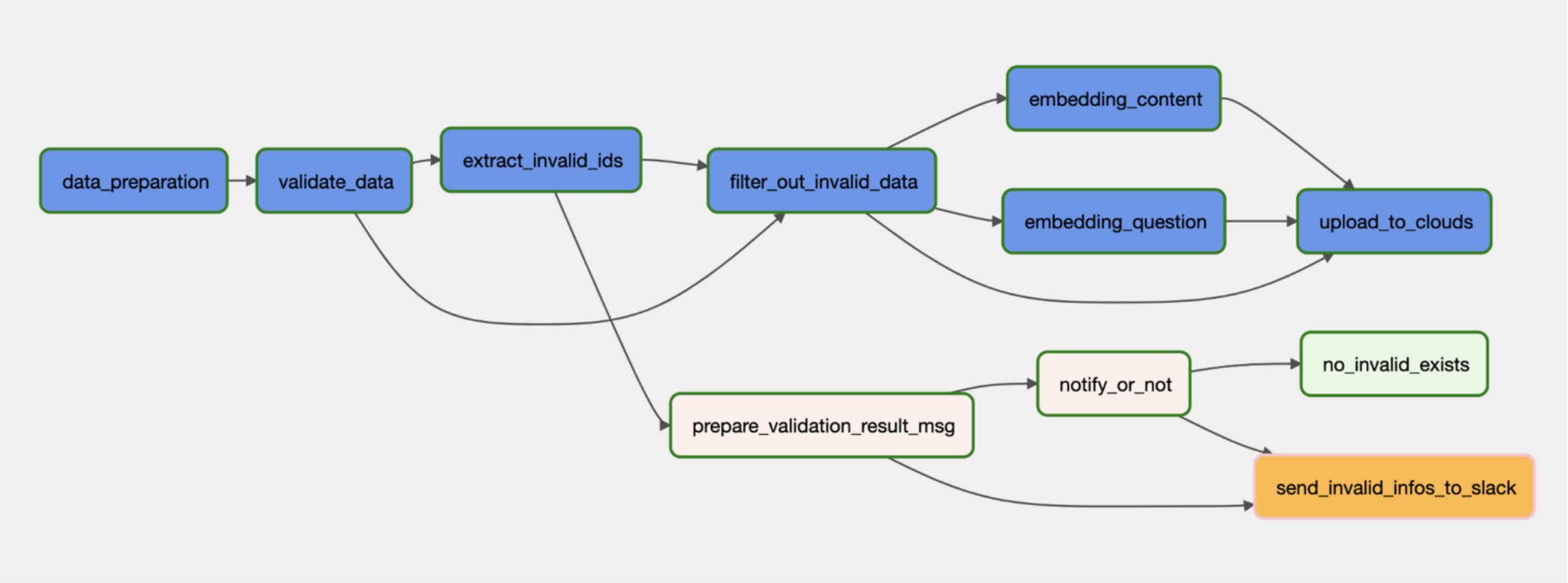
<After>



Before & After

DEVIEW 2021

<After>



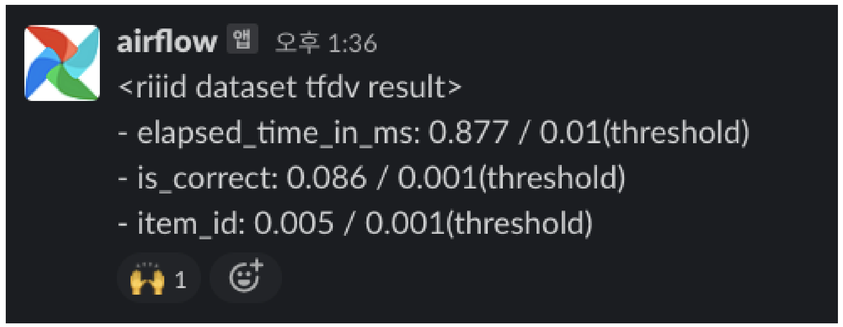
High cost job 관리


```
tfdv_operator = BeamRunPythonPipelineOperator(
    task_id="run_tfdv_on_dataflow",
    py_file='beam_pipeline_tfdv.py',
    pipeline_options={
        "date_time_folder": RUNNING_TIME,
        "train_stats_output_path": f"gs://tfdv/{RUNNING_TIME}/train.tfrecord",
        "test_stats_output_path": f"gs://tfdv/{RUNNING_TIME}/test.tfrecord",
    },
    py_requirements=[
        "apache-beam[gcp]==2.31.0", "tensorflow-data-validation==1.1.0",
    ],
    runner="DataflowRunner",
    dataflow_config=DataflowConfiguration(
        job_name="{{task.task_id}}",
        location="asia-northeast3",
    ),
    default_pipeline_options={
        "temp_location": "gs://tfdv/dataflow_job_tmp/",
        "staging_location": "gs://tfdv/dataflow_job_staging/",
    },
)
```

TensorFlow Extended

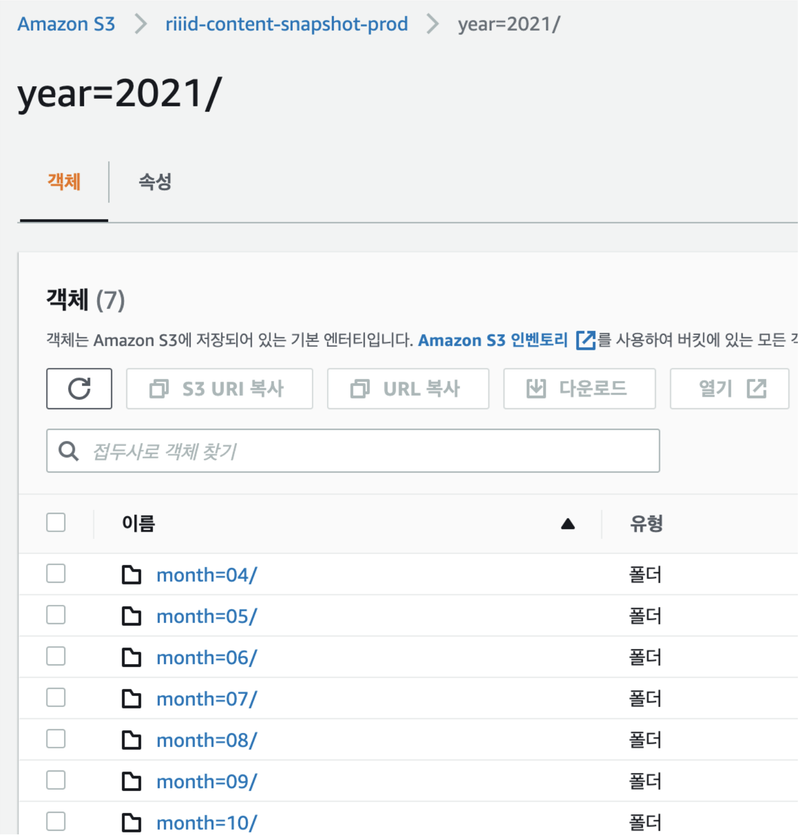
Apache Beam

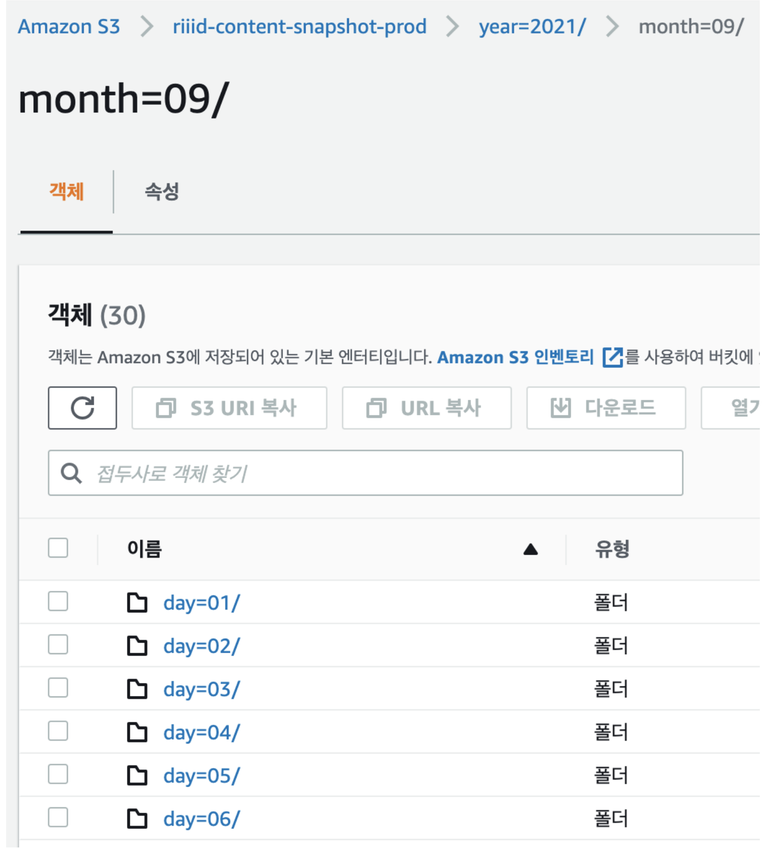
Google Cloud Dataflow

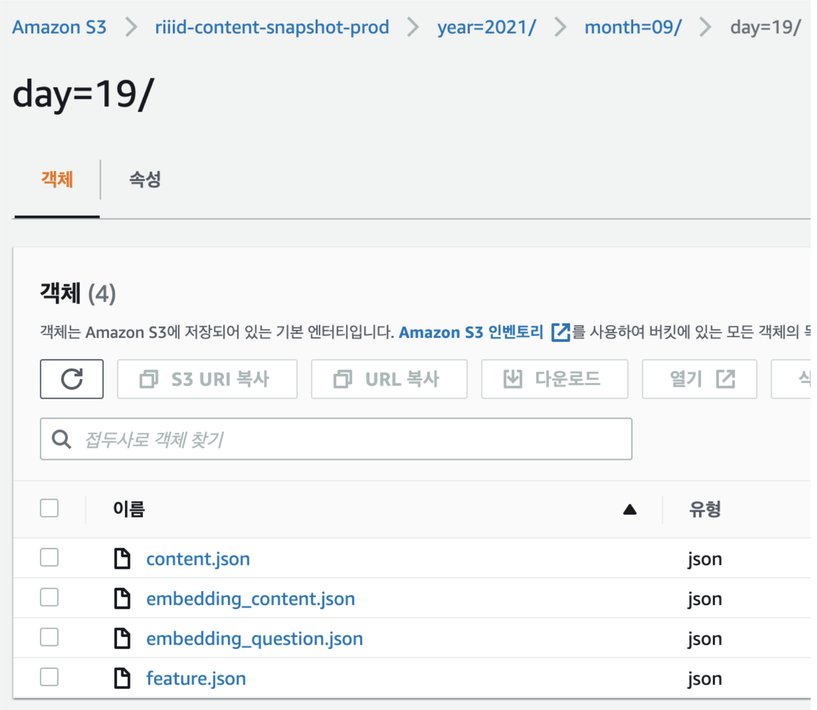


Load data into Cloud Data Warehouse

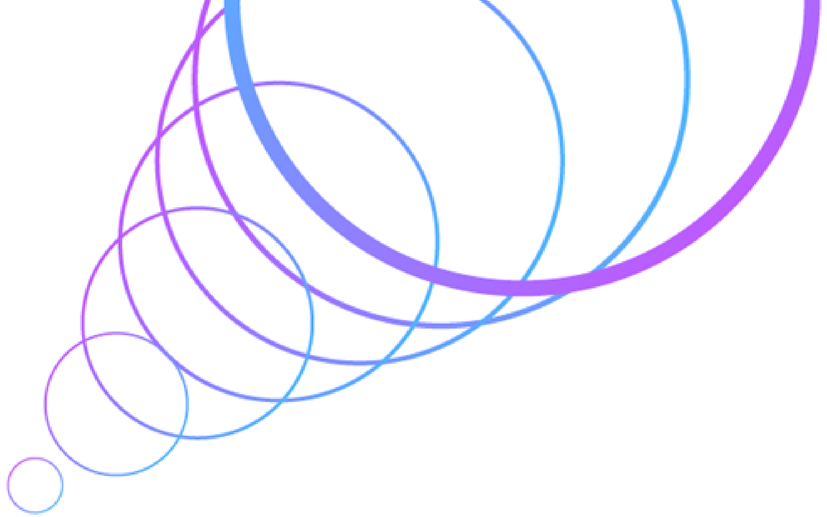
DEVIEW 2021







매일마다 그 날의 Content 정보를 기록한 Content Snapshot 파일이 업로드



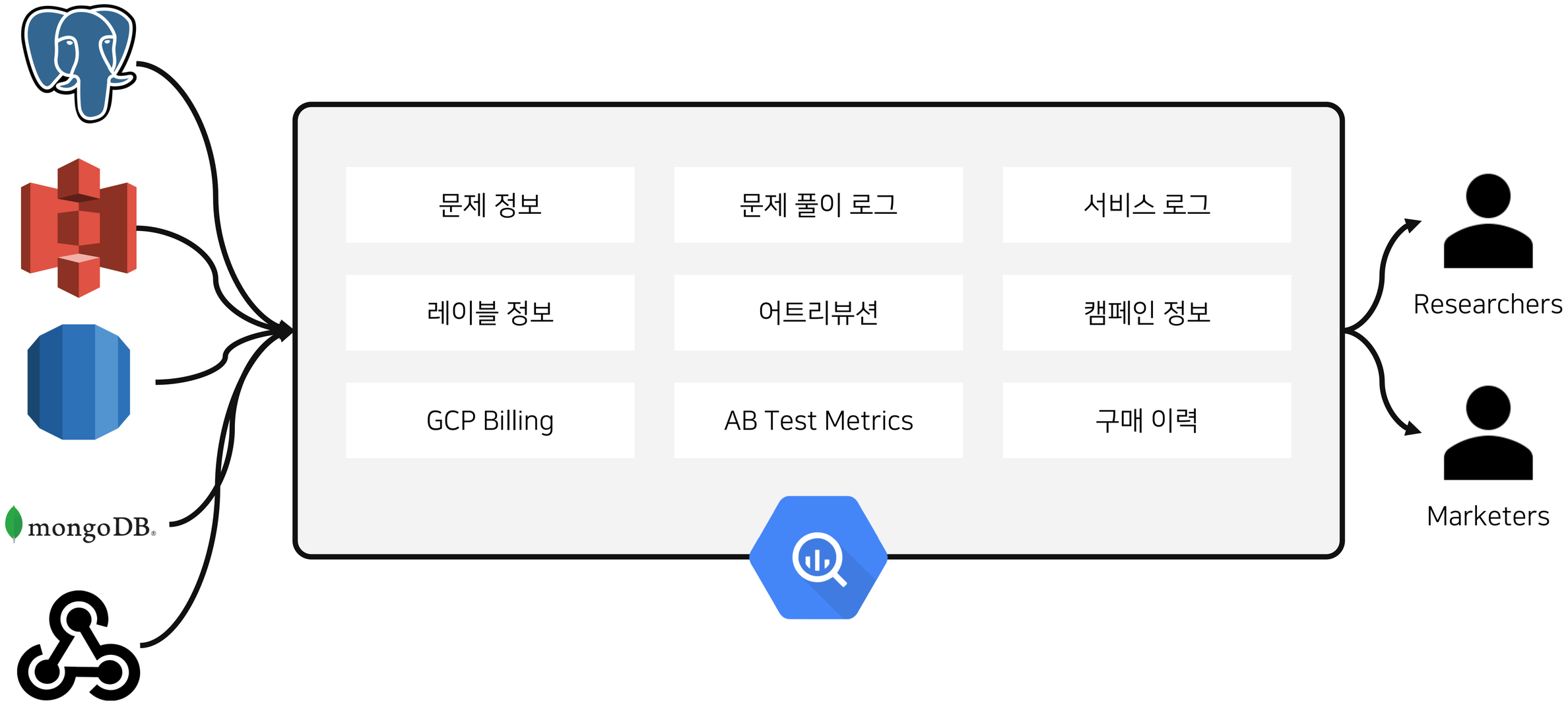
DEVIEW 2021

Riiid Tutor의 데이터를 흐르게 하는 기술

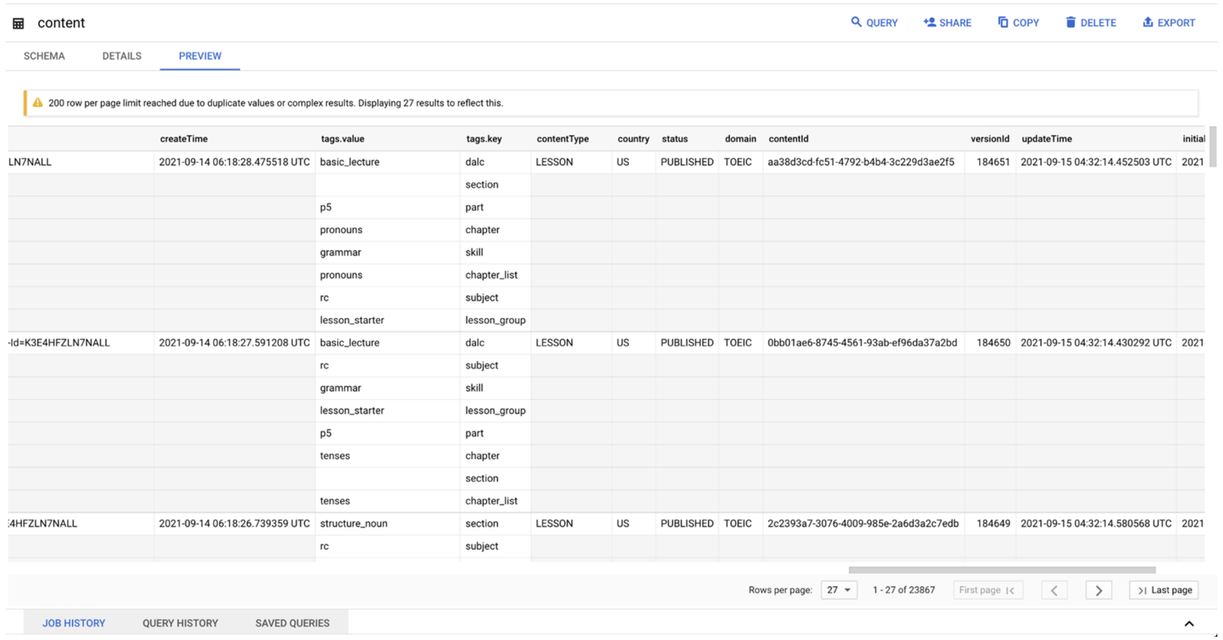
한성민, 최정우 Riiid

4.BigQuery 비정형 데이터 수집과 유연성 한번에 챙기기

BigQuery


mongoDB.
문제 정보
문제 풀이 로그
서비스 로그
레이블 정보
어트리뷰션
캠페인 정보
GCP Billing
AB Test Metrics
구매 이력
Researchers
Marketers

GCP + Terraform



BigQuery Dataset 단위를
tfstate로 형상화하여 관리

```hcl
resource "google_bigquery_dataset" "dataset_name" {
  dataset_id = "dataset_id"
  location   = "US"

  labels = merge(local.common_labels, {
    name        = "resource_name"
    environment = "all"
  })

  access {
    role         = "OWNER"
    special_group = "projectOwners"
  }
  access {
    role         = "READER"
    special_group = "projectReaders"
  }
  access {
    role         = "WRITER"
    special_group = "projectWriters"
  }
}
```

적재 데이터의 문제

DEVIEW 2021

Content 데이터

```
{
    "contentId": "bdd8b79a-8137-4e9e-91df-d41e54565a4a",
    "contentType": "QUESTION_SET",
    "contentBody": {
        "questionSet": {
            "questions": [ {
                "description": "...",
                "explanation": "...",
                "objective": { ... }
            }]
        }
    },
    "domain": "TOEIC",
    ...
    "tags": [
        {
            "key": "difficulty",
            "value": { "stringValue": "hard" }
        }
    ],
    "versionId": 233195
}
```

```
oneof state {
    QuestionSetState question_set_state = 8;
    QuestionState question_state = 9;
    LessonSetState lesson_set_state = 10;
    LessonState lesson_state = 11;
    VocaState voca_state = 14;
}
```

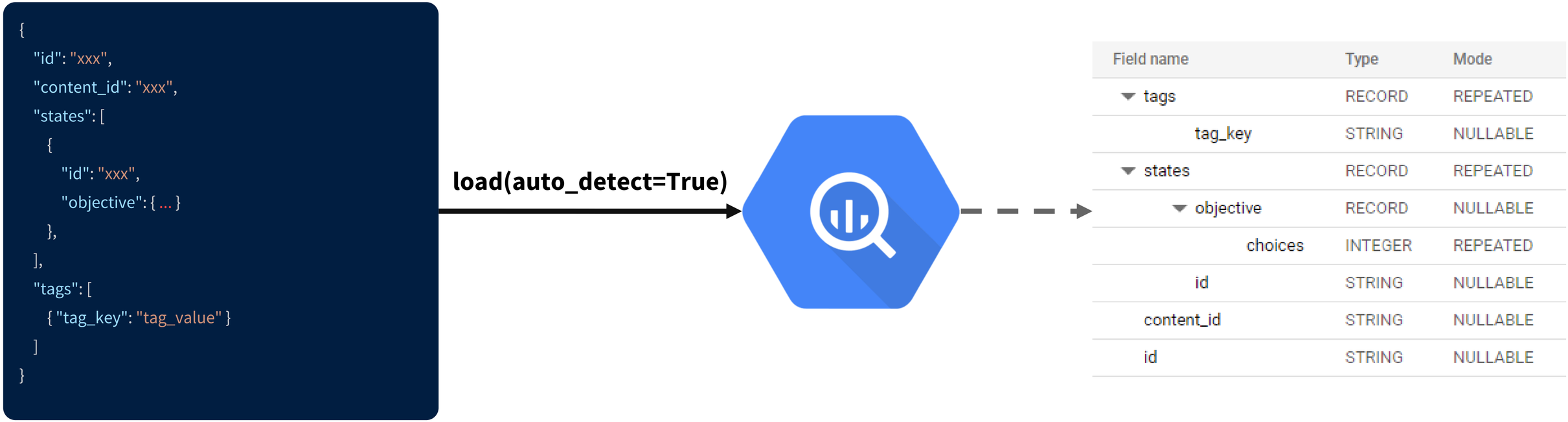
protobuf3 스키마를 기반하여 로깅 시스템이 구성

이로 인해 로그 포맷의 관리는 용이하나

정형화를 위한 데이터 엔지니어링의 난이도 ↑

BigQuery schema auto-detection

DEVIEW 2021



JSON 포맷을 파싱하여 정형화된 BigQuery 스키마로 자동 적용

https://cloud.google.com/bigquery/docs/schema-detect

Record column

DEVIEW 2021

```
{
  "tags": {
    "chapter_list": ["detail", "work", "subject"]
  }
}
```

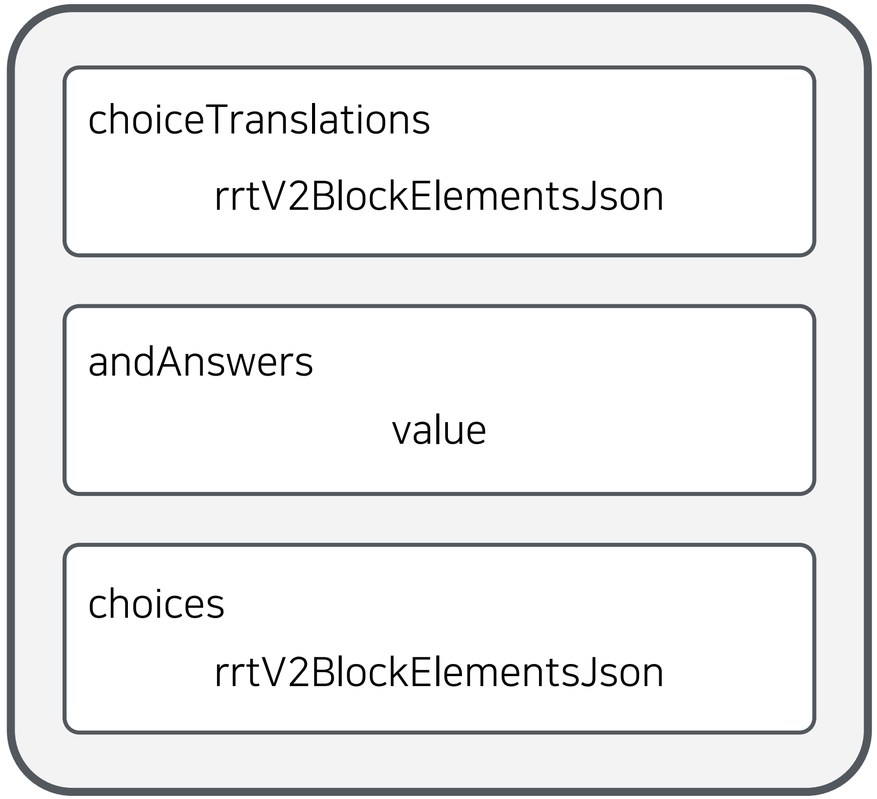
Field name	Type	Mode
tags	RECORD	NULLABLE
chapter_list	STRING	REPEATED

```
{
  "states":[
    {
      "questionSet":{
        "questions":[{  "id":"question1", "description":"question1_description" }]
      }
    },
    {
      "question":{
        "id":"question1",
        "description":"question1_description"
      }
    }
  ]
}
```

```
oneof state {
  QuestionSetState questionSet = 8
  QuestionState question = 9
}
```

Field name	Type	Mode
states	RECORD	REPEATED
question	RECORD	NULLABLE
description	STRING	NULLABLE
id	STRING	NULLABLE
questionSet	RECORD	NULLABLE
questions	RECORD	REPEATED
description	STRING	NULLABLE
id	STRING	NULLABLE

Record column 예시



Question.objective

▼ objective	RECORD	NULLABLE
▼ choiceTranslations	RECORD	REPEATED
rrtV2BlockElementsJson	STRING	NULLABLE
▼ andAnswers	RECORD	NULLABLE
value	INTEGER	REPEATED
▼ choices	RECORD	REPEATED
rrtV2BlockElementsJson	STRING	NULLABLE

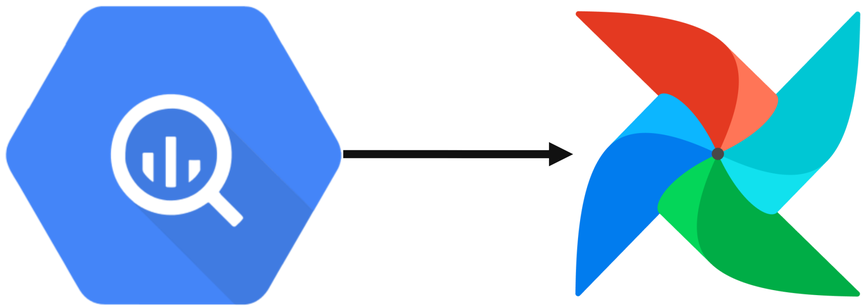
Riiid의 정답 스키마 적용 예시

Query & Flatten


```
WITH T AS (
  SELECT q.id AS item_id, SUBSTRING(q.pack_type, 11, 12) AS part, exp.correct_answer,  q.locale
  FROM `riiid-research.content_repo_prd.questions` AS q
  LEFT JOIN `riiid-research.content_repo_prd.explanations` AS exp
    ON q.id=exp.question_id
  LEFT OUTER JOIN `riiid-research.content_repo_prd.explanations` AS t_exp
    ON exp.question_id=t_exp.question_id AND exp.updated_at < t_exp.updated_at
  LEFT JOIN `riiid-research.content_repo_prd.labels` AS label
    ON q_label.label_id=label.id
  WHERE t_exp.question_id IS NULL
  GROUP BY
    q.id, q.pack_type, q.pack_id, q.locale,
    q.created_at, q.updated_at, exp.correct_answer
)
SELECT item_id, part, ARRAY_TO_STRING(ARRAY(SELECT CAST(x AS STRING) FROM UNNEST(label_ids) AS x ORDER BY x), ';') AS tags,
correct_answer, locale
FROM T
```

빅쿼리를 데이터 셋 파이프라인을 통해 모델 학습 데이터 가공 예시

BigQuery with Pipeline



Airflow BigQueryOperator 및 XCom을 통한 데이터 파이프라인 구현

```
get_question_and_features_from_bq = BigQueryInsertJobOperator(
  task_id="get_question_and_features_from_bq",
  configuration={
    "query": {
      "query":
get_query("get_question_and_features_from_bq.sql"),
      "useLegacySql": False,
      "destinationTable": {
        "projectId": "project-name",
        "datasetId": GCP_RESEARCH_DATASET,
        "tableId": "questions",
      },
      "writeDisposition": "WRITE_TRUNCATE",
      "allowLargeResults": True,
    }
  },
  location=BIGQUERY_LOCATION,
)
```

BigQuery 적용 후 이점

DEVIEW 2021

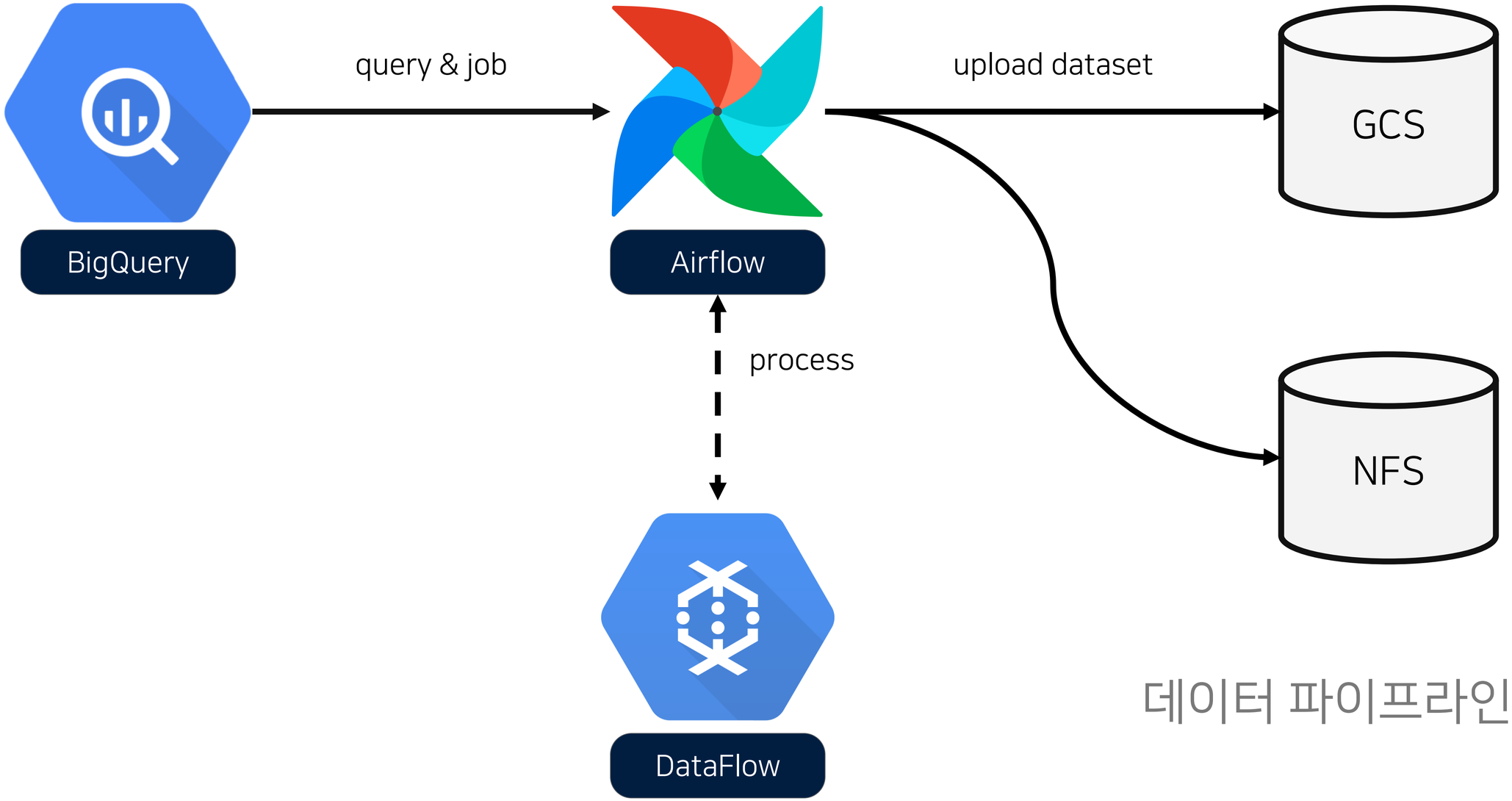
변화하는 Feature에 대응이 유연하다.

- Feature 구조는 서비스의 기획과 설계에 따라 수시로 변한다.
  EX) 문제만 제공하던 서비스에 단어장 기능 추가, 컨텐츠 스팩 변경
- auto detect schema를 사용하면 스키마 변경에 따른 엔지니어 부하가 줄어든다.

리서쳐가 필요할 때 원하는 데이터를 만들 수 있다.

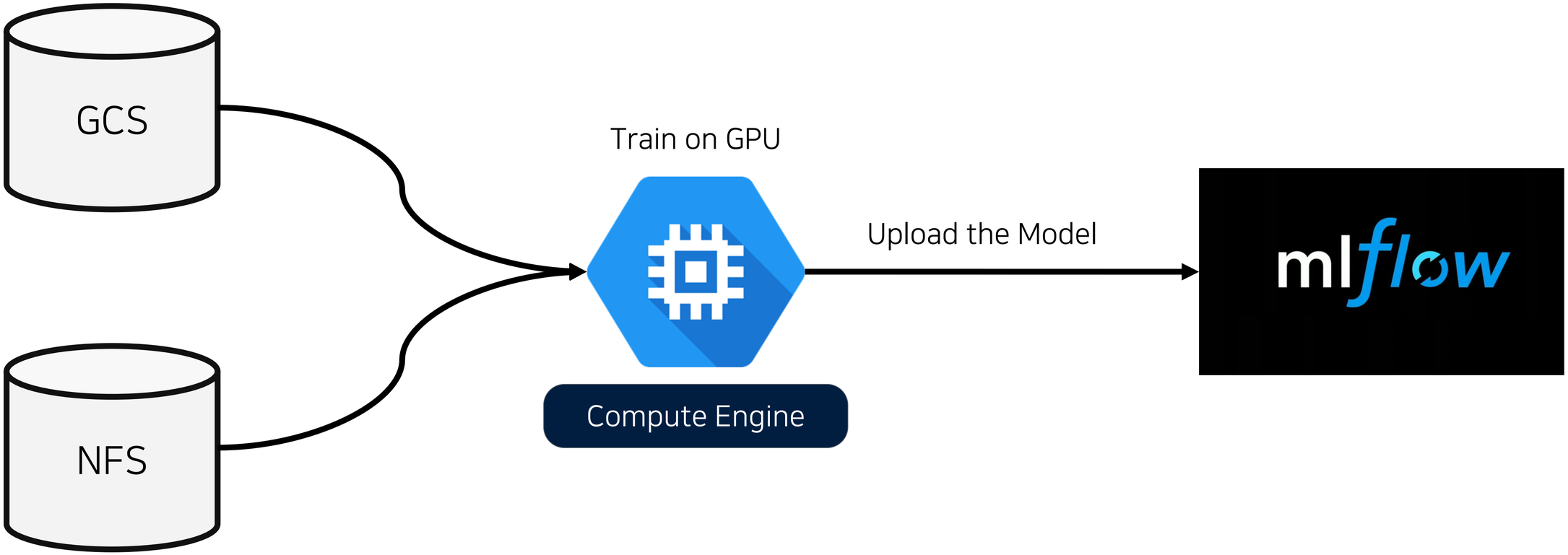
- 일반적인 다른 데이터 웨어하우스는 추가 데이터 가공에 데이터 엔지니어에게 요청이 필요하고, 자칫 전체 파이프라인이 Fail 될 수 있다.
- BigQuery의 경우 모든 데이터를 적재하고, 리서치가 쿼리를 통해 직접 데이터를 가공할 수 있도록 하면 이 문제가 해결되며 BigQuery scan은 스캔 범위에 따라 Worker를 유동적으로 할당하므로 빠른 가공 속도를 제공한다.

Use case: Dataset processing



데이터 파이프라인 구성

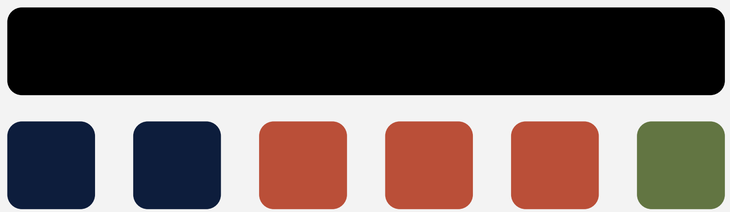
Use case: ML training using dataset



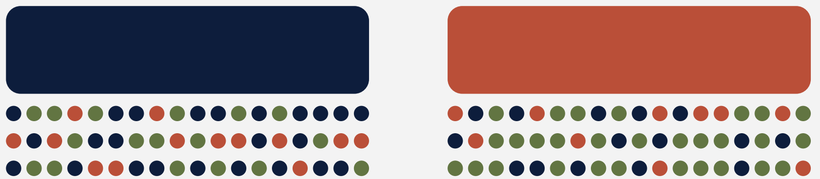
가공한 데이터셋을 통한 Model Training & Upload

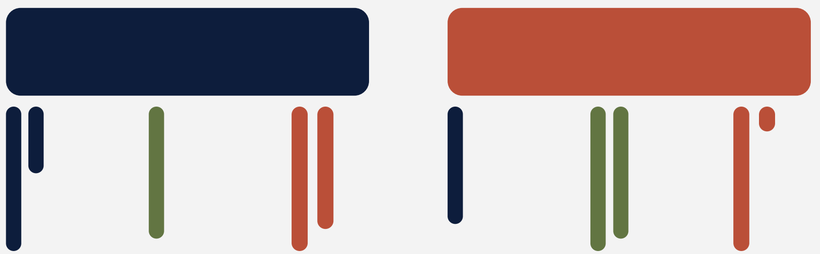
BigQuery 요금 최적화

파티셔닝



클러스터링





파티셔닝 vs 클러스터링

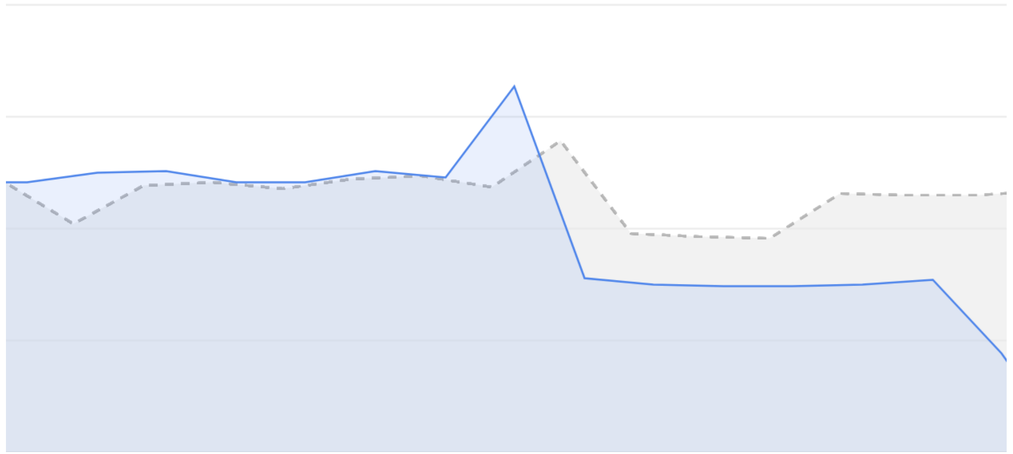
파티셔닝 (Partitioning)

- 테이블을 파티션 단위로 명확히 나누어 관리할 수 있음.
- 쿼리를 실행하기 전 쿼리 예상 비용을 비교적 정확하게 예측할 수 있음.
- 단일 키에 대한 적용이 가능.

클러스터링 (Clustering)

- 열, 열 그룹의 카디널리티가 비교적 큰 경우 사용.
- 쿼리를 실행하기 전 쿼리 예상 비용을 정확하게 예측하기 어려움.
- 1개 이상의 키에 대한 적용이 가능.

최적화 사례: Riiid



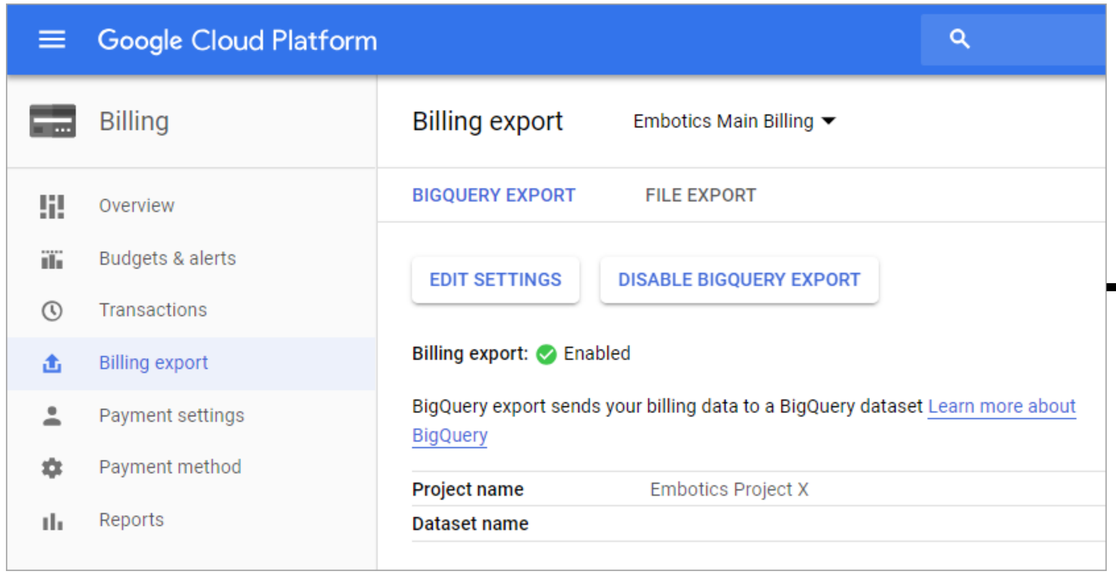
테이블 특성에 차이가 존재하지만
약 40~60% 절감

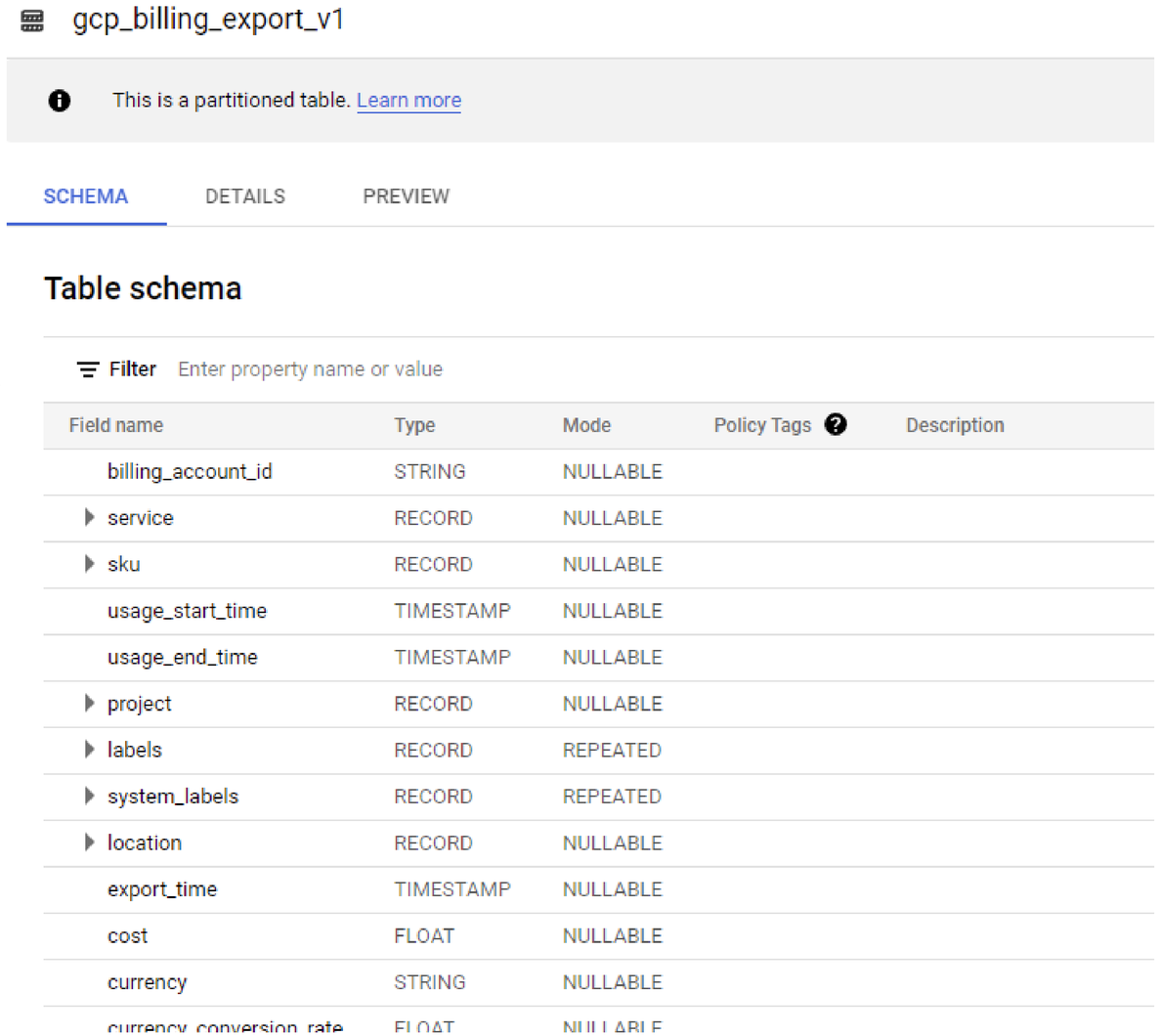
```
bq cp \
    sample_dataset.sample_table \
    sample_dataset.sample_table_copy
```

```
CREATE TABLE sample_dataset.sample_table
PARTITION BY DATE(partition_column)
AS SELECT * FROM sample_dataset.sample_table_copy;
```

```
DROP TABLE sample_dataset.sample_table_copy
```

BigQuery Billing Report



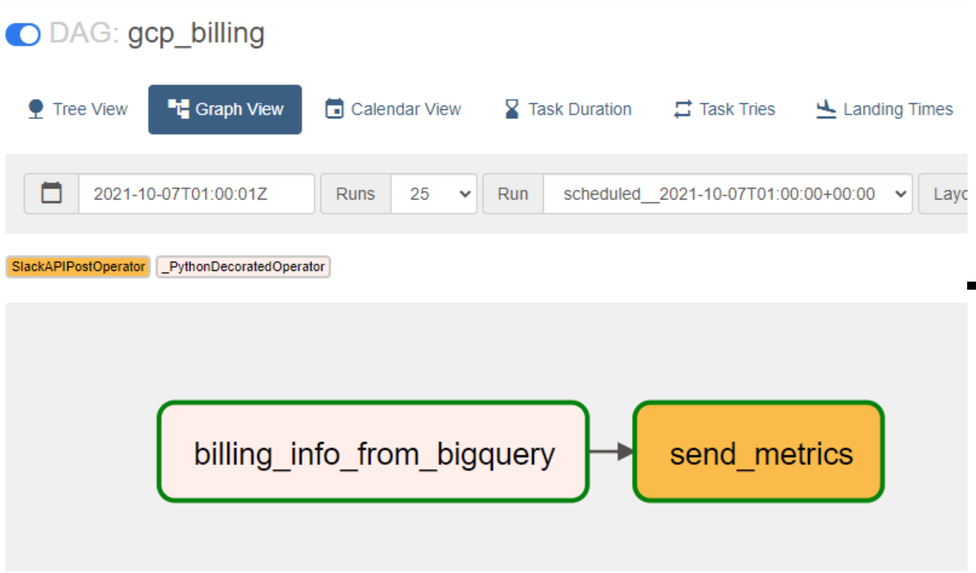


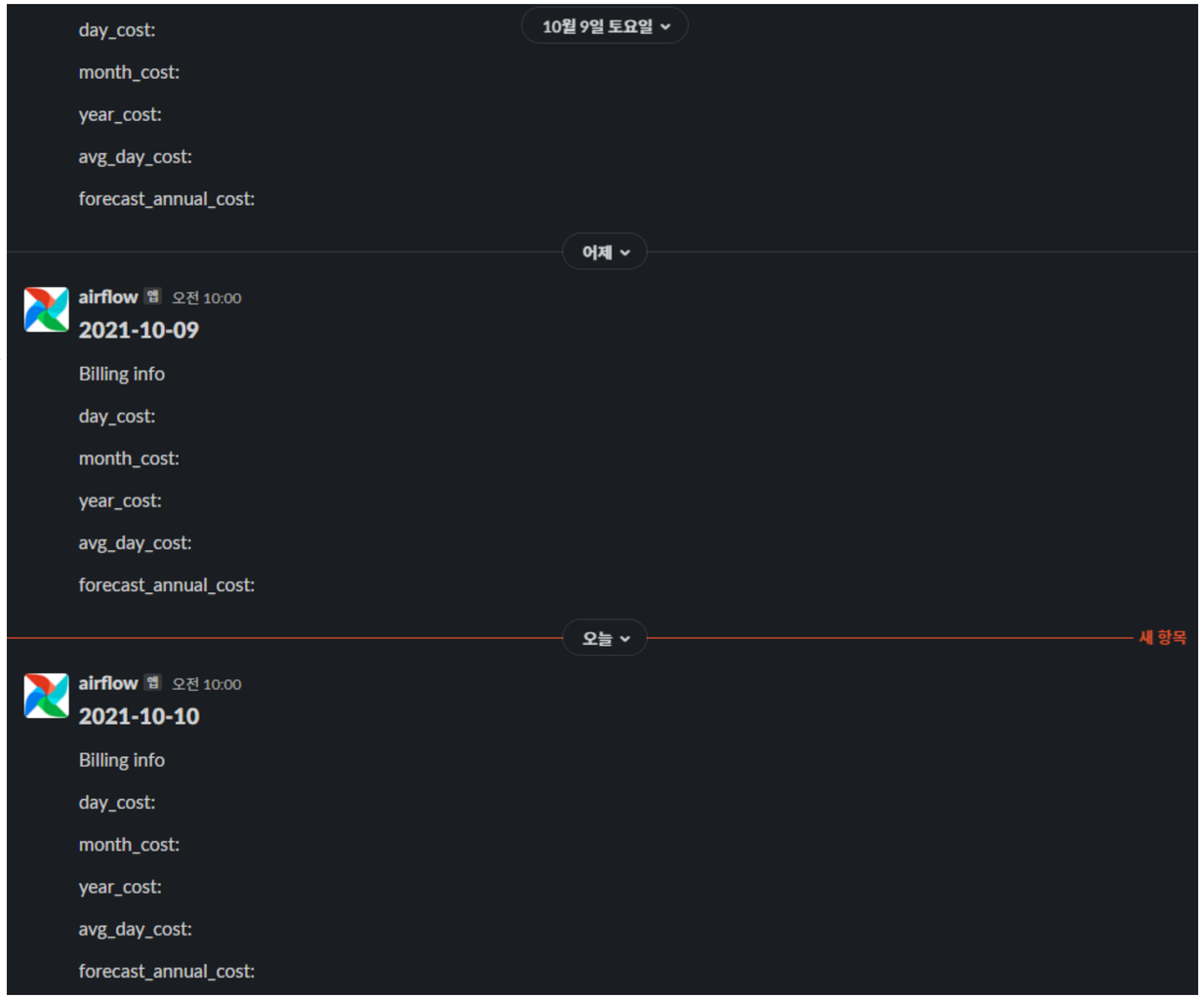
GCP Billing Report를
BigQuery Dataset으로 export

https://cloud.google.com/billing/docs/how-to/reports

Billing monitoring 파이프라인 구성

DEVIEW 2021





Airflow DAG를 통한
Billing report monitoring 파이프라인

5. 결론

결론

컨텐츠가 업데이트 될 때 마다 주기적으로 서빙 시스템에 반영할 수 있을까?

데이터 형태가 바뀔 때 Side-effect 추정을 할 수 있을까?.

수 많은 데이터 소스의 비정형 데이터를 보관하고 연구에 사용 가능할까?

데이터 웨어하우스 운영에 비용을 절감할 순 없을까?

결론

컨텐츠가 업데이트 될 때 마다 주기적으로 서빙 시스템에 반영할 수 있을까?
가능하다. => Airflow Scheduling & Triggers, Smart sensors

데이터 형태가 바뀔 때 Side-effect 추정을 할 수 있을까?.
가능하다. => Airflow2 TaskFlow API

수 많은 데이터 소스의 비정형 데이터를 보관하고 연구에 사용 가능할까?
가능하다. => BigQuery auto detect schema

데이터 웨어하우스 운영에 비용을 절감할 순 없을까?
가능하다. => Partitioning & Clustering, GCP Billing Report Monitoring

결론

컨텐츠가 업데이트 될 때 서빙에 어떻게 반영하지?

- Answer is Airflow schedule job

Airflow DAG 구성을 통해 Daily, Weekly 데이터 파이프라인 구성이 가능
데이터 변경에 대한건 Smart sensor를 통해서 감지하고, 파이프라인 실행이 가능

결론

DEVIEW 2021

데이터 형태가 바뀔 때 Side-effect 추정이 불가능하다.

- Answer is Airflow2 TaskFlow API

Airflow 2 TaskFlow 스팩에서는 DAG 구성에 Task의 흐름이 곧 데이터의 흐름을 의미한다.
따라서 Task 중 일부의 데이터 형태가 바뀌더라도 그로 인해 어떤 다른 Task들이 영향을 받을지 알고 대응할 수 있다.

결론

수 많은 데이터 소스의 비정형 데이터를 어떻게 보관하지?

- Answer is schema auto-detecting of BigQuery

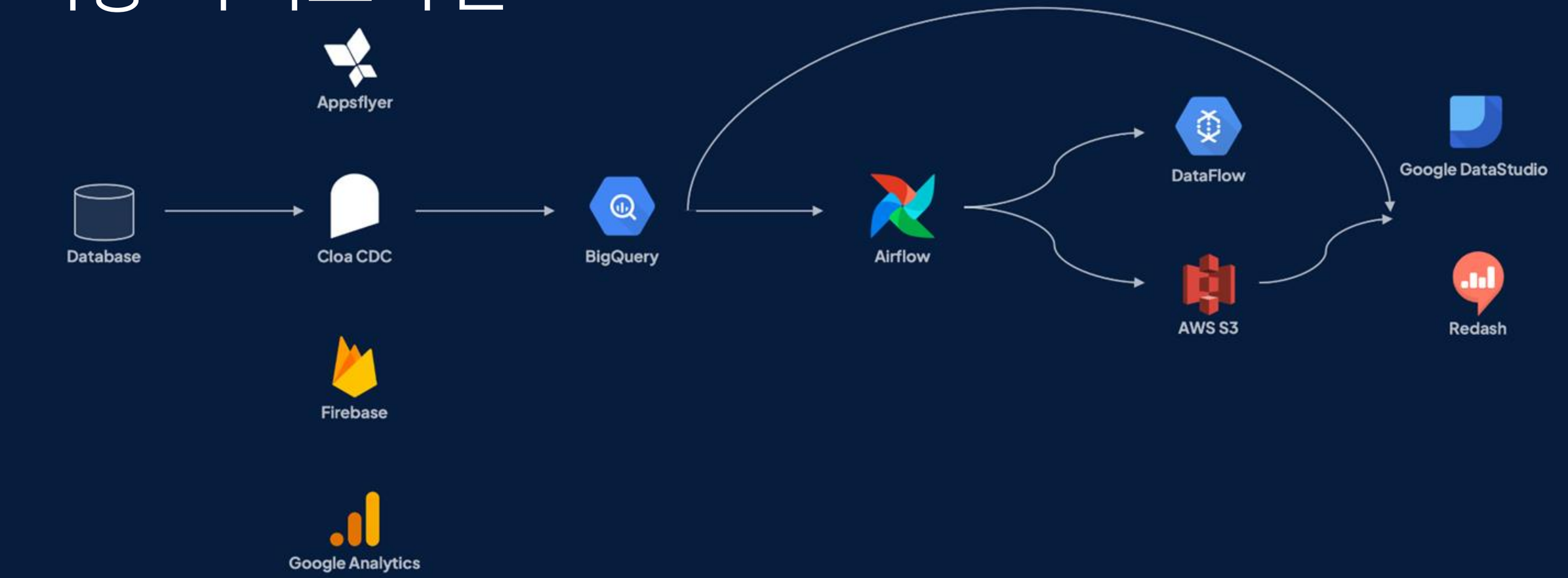
- BigQuery는 정형으로 데이터를 보관하지만, 데이터를 로드하는 과정에서 JSON 스키마를 파싱하고 그에 맞는 스키마를 감지할 수 있다.
- 또한 Complex struct 컬럼 타입을 지원하기 때문에 복잡한 문제 정보를 보관하고 연구용으로 사용하기 알맞다.

결론

데이터 웨어하우스 운영에 비용을 절감할 순 없을까?

- BigQuery는 기본적으로 정량 요금제이기 때문에 사용한 만큼 비용을 지불한다.
- Table partitioning & clustering을 통해 요금을 최적화 할 수 있다.
- GCP Billing Report를 통해 비용 모니터링 파이프라인 구성을 고민해보자.
- 너무 많은 스캔이 필요한 데이터셋은 Slot 계약을 고민하자.

최종 파이프라인


Appsflyer
Database
Cloa CDC
BigQuery
Airflow
DataFlow
Google DataStudio
AWS S3
Redash
Firebase
Google Analytics
Database
Data source
Data warehouse
Data workflow
Data destination & proceesor
Report

Join us!

DEVIEW 2021

함께 문제를 해결해나가고 싶은 분이라면

https://riiid.com/ko/career

Thank You