실시간 추천 시스템을 위한 Feature Store 구현기

이영수
HYPERCONNECT

NAVER DEVIEW 2023

CONTENTS

1. 하이퍼커넥트의 추천 시스템
2. 왜 Feature Store를 도입했는가?
3. 하이퍼커넥트의 Feature Store
4. 적용 사례 & 도입 효과

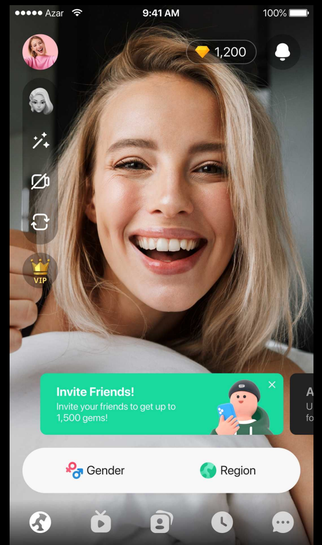
1. 하이퍼커넥트의 추천 시스템

1.1 하이퍼커넥트의 추천 시스템

NAVER
DEVIEW
2023

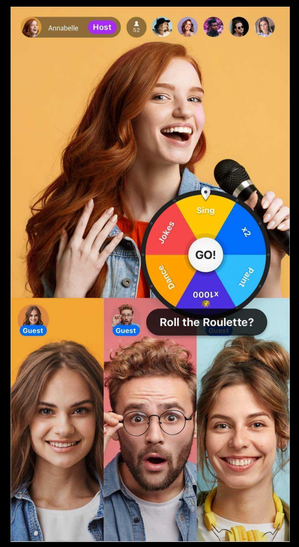
azar

전 세계 사람들과 스와이프 한번으로 연결하는 소셜 디스커버리 서비스



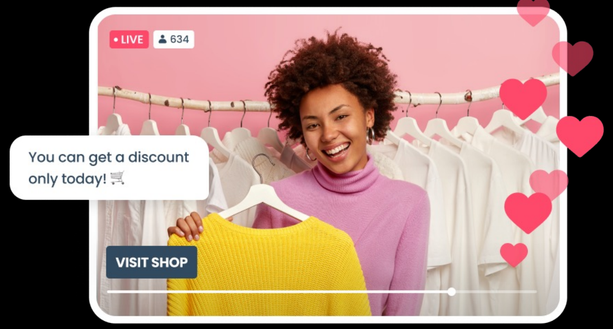
HAKUNA

누구나 자유롭게 방송에 참여할 수 있는 인터랙티브한 소셜 라이브 스트리밍



HYPERCONNECT
ENTERPRISE

하이퍼커넥트의 비디오 기술이 축적된 B2B 솔루션



1.1 하이퍼커넥트의 추천 시스템

azar

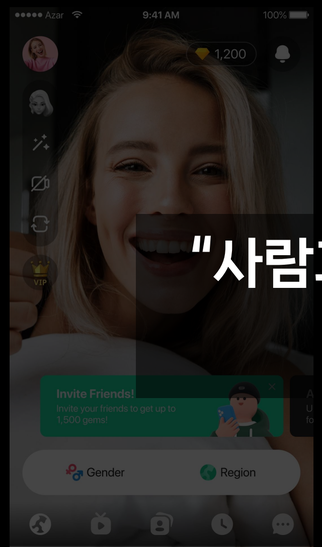
- 1:1 매칭 대상 추천
- 라운지 카드 추천
- 라이브 방송 추천

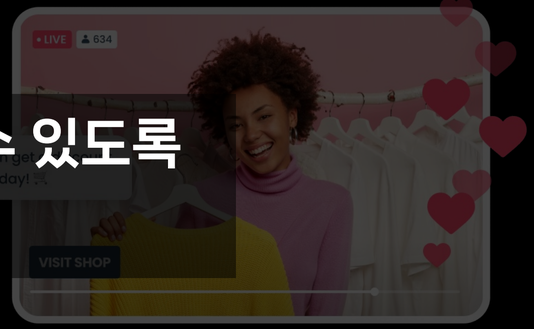
HAKUNA

- 라이브 방송 추천
- 인기 스트리머 추천

HYPERCONNECT ENTERPRISE

- 라이브 방송 추천
(B2B 솔루션 내장)





"사람과 사람을 연결하면서 유저들에게 더 나은 경험을 줄 수 있도록
다양한 기능에 추천 시스템 운영 중"

1.2 다른 추천 시스템과의 차이점

NAVER
DEVIEW
2023

아이템 = 유저

- 상품, 콘텐츠 등 정적인 아이템을 추천해 주는 일반적인 추천 시스템과 달리,
  유저를 추천해 주는 특별한 도메인

실시간 추천 시스템

- 하이퍼커넥트의 추천 시스템은 대부분 온라인 유저만 추천할 수 있다는 제약 존재
- 더불어 유저 – 유저 추천 시스템에서는 두 유저가 모두 신규(Cold)인 경우에
  실시간 데이터(실시간 행동 로그, 컨텍스트 등)를 사용하지 않으면 매우 낮은 추천 성능을 보일 수 있음
- 따라서 실시간성을 잘 고려한 추천 시스템이 필수

2. 왜 Feature Store를 도입했는가?

2.1 추천 시스템과 Feature

NAVER DEVIEW 2023

Dataset

<User features>					<Item (peer user) features>					<Target>
id	gender	country	age	avg_chat_sec	id	gender	country	age	avg_chat_sec	chat duration
1001	MALE	KR	20	12	2001	FEMALE	KR	21	3	142
1001	MALE	KR	20	12	2002	FEMALE	JP	23	3	5
1002	MALE	JP	26	142	2002	FEMALE	JP	25	71	35
1002	MALE	JP	26	142	2003	FEMALE	CA	21	11	51
1003	FEMALE	US	22	48	2001	FEMALE	KR	21	3	11
1003	FEMALE	US	22	48	2002	FEMALE	JP	23	71	121
1003	FEMALE	US	22	48	2003	FEMALE	CA	25	11	26

Target (chat duration)을 예측하도록 학습

ML/AI 모델

Unknown Input (Online Environment)

1001	MALE	KR	20	12	2003	FEMALE	CA	25	11	???

유저 1001이 유저 2003를 만나면 대화시간이 얼마나 나올지 예측 (추론)

2.1 추천 시스템과 Feature

NAVER DEVIEW 2023

Dataset

<User features>					<Item (peer user) features>					<Target>
id	gender	country	age	avg_chat_sec	id	gender	country	age	avg_chat_sec	chat duration
1001	MALE	KR	20	12	2001	FEMALE	KR	21	3	142
1001	MALE	KR	20	12	2002	FEMALE	JP	23	3	5
1002	MALE	JP								
1002	MALE	JP								
1003	FEMALE	US								
1003	FEMALE	US	22	48	2002	FEMALE	JP	23	71	121
1003	FEMALE	US	22	48	2003	FEMALE	CA	25	11	26

온라인 모델 추론을 위해서는 user/item에 해당하는 feature를 온라인에서 불러올 수 있어야 함

Target (chat duration)을 예측하도록 학습

ML/AI 모델

Unknown Input (Online Environment)

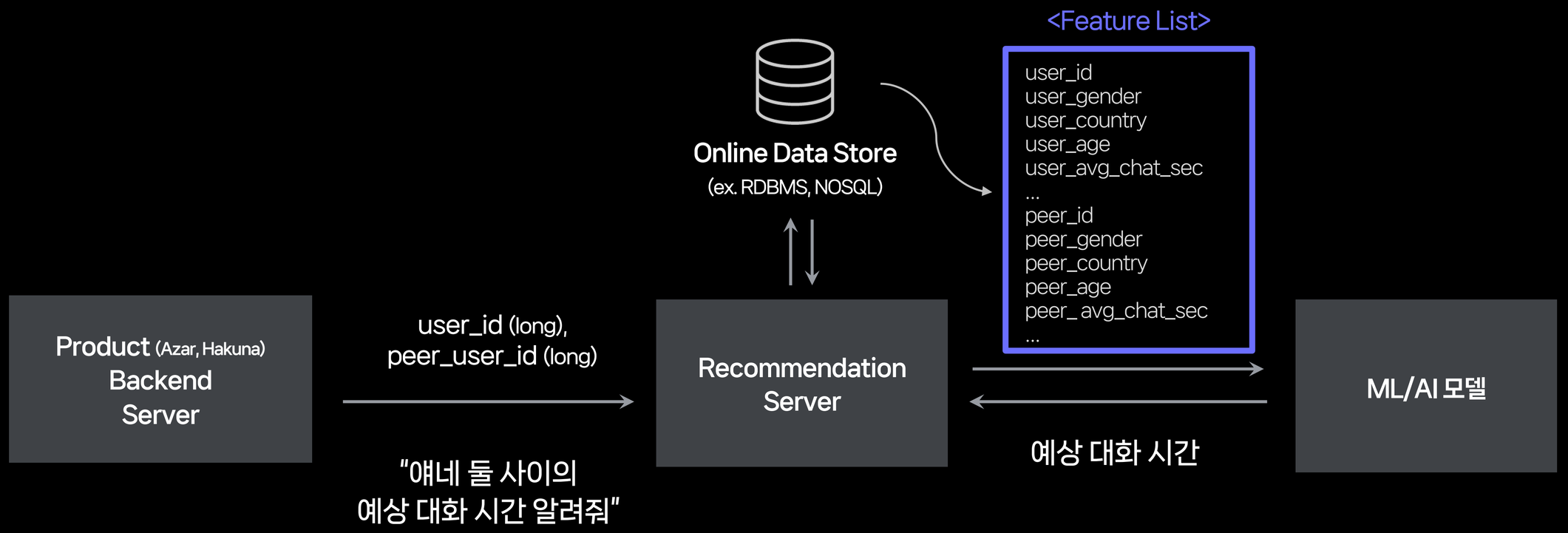
1001	MALE	KR	20	12		2003	FEMALE	CA	25	11		???

유저 1001이 유저 2003를 만나면 대화시간이 얼마나 나올지 예측 (추론)

2.1 추천 시스템과 Feature

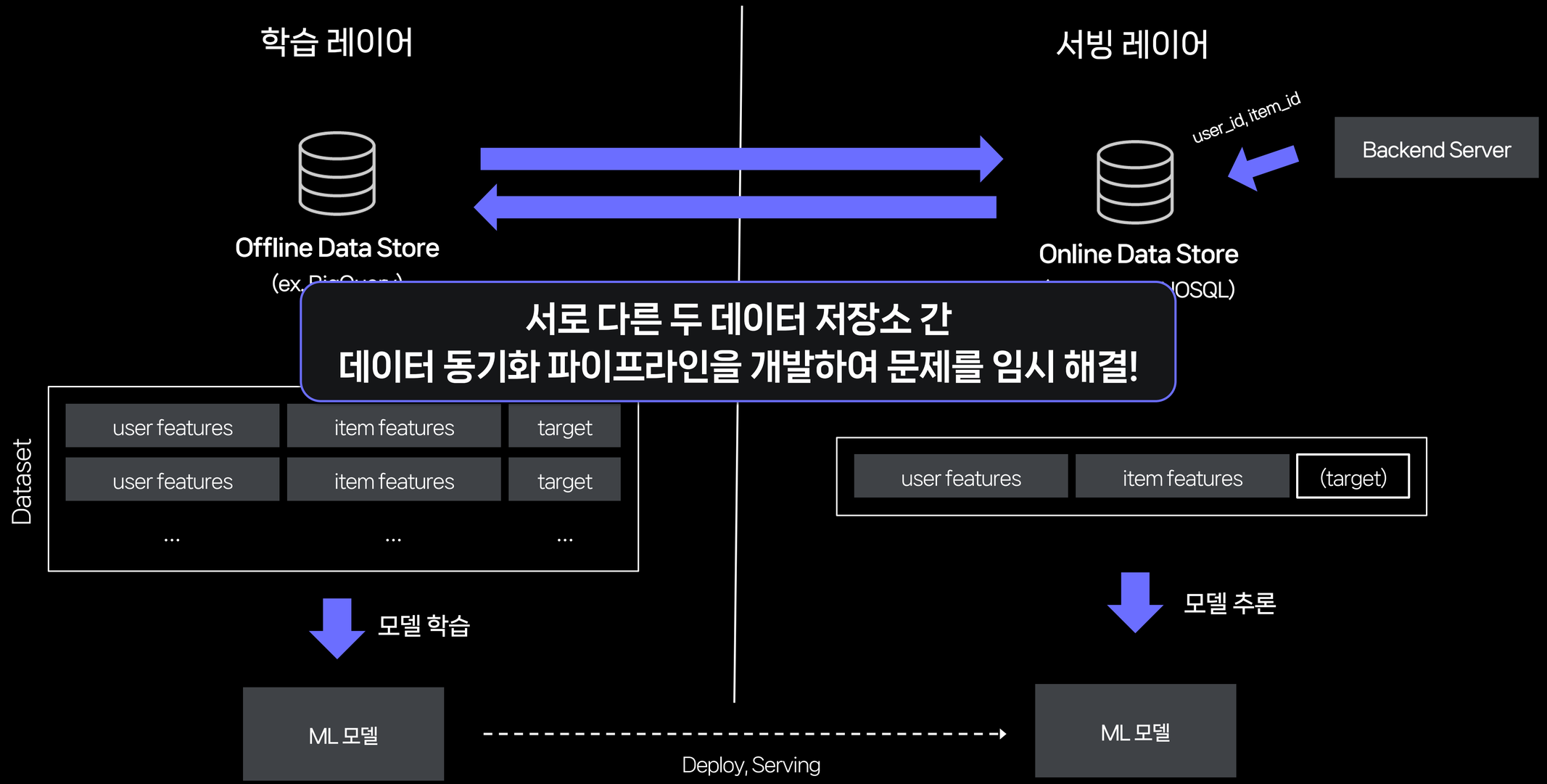
NAVER
DEVIEW
2023

- 온라인 추천을 위해서는 Feature 데이터를 불러올 수 있는 데이터 저장소가 온라인 환경에서도 필요
- 문제: 같은 Feature에 대해서 오프라인(BigQuery), 온라인(RDBMS, NOSQL) 저장소 모두에 저장되어야 함



2.2 Feature Store 이전의 하이퍼커넥트 추천 시스템

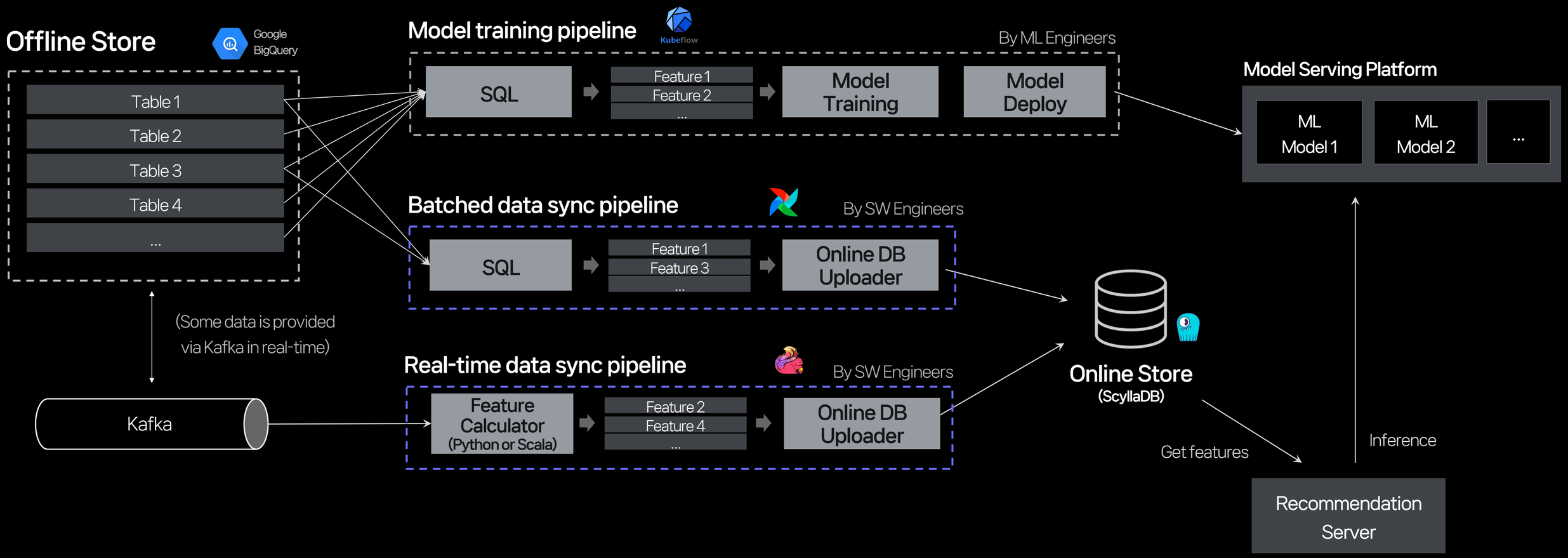
NAVER DEVIEW 2023



2.2 Feature Store 이전의 하이퍼커넥트 추천 시스템

NAVER DEVIEW 2023

- 실제 시스템 아키텍처는 아래와 같은 구조를 가졌었음



2.3 겪은 문제들

NAVER
DEVIEW
2023

- 하나의 추천 시스템을 운영할 땐 Feature Store가 없어도 문제가 크지는 않았음
- 하지만 다양한 곳에 추천 시스템을 적용하며 더 많은 문제가 발생

01 **학습/서빙 데이터 불일치**

02 **피쳐 추가 시 높은 엔지니어링 비용**

03 **다수의 추천 시스템 운영 시 컴포넌트 중복**

04 **여러 추천 시스템 간 피쳐 공유의 어려움**

2.3 겪은 문제 1 - 학습/서빙 데이터 불일치

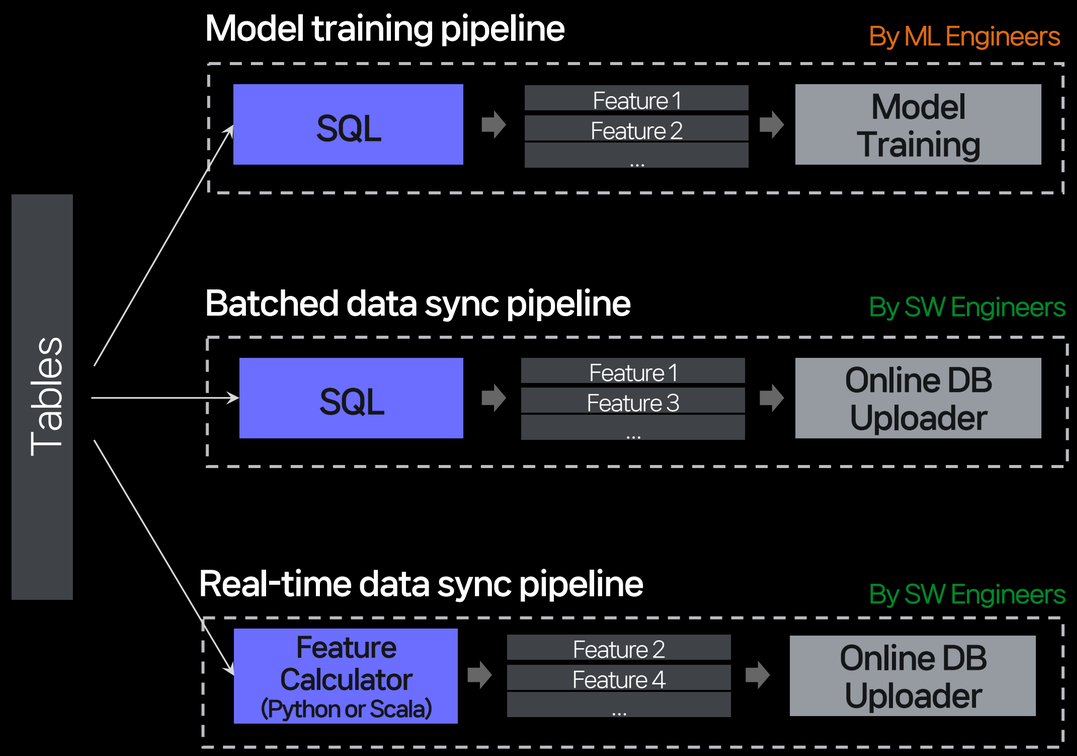
NAVER DEVIEW 2023

Ideal:

같은 User ID로 피쳐를 조회하면 학습 & 서빙 레이어 모두에서 같은 데이터 반환

Reality:

- 피쳐 계산 로직이 3군데에 나누어져 있음
- 파이프라인마다 엔지니어가 다름
- → 학습/서빙 레이어 간 데이터 불일치 문제 발생
  (ex. 같은 유저에 대한 avg_chat_duration / time spent 값이 BigQuery와 Online DB 사이에 서로 다름)



2.3 겪은 문제 2 - 피쳐 추가 시 높은 엔지니어링 비용

NAVER
DEVIEW
2023

모델에 새로운 피쳐를 추가하기 위한 *SW 엔지니어링* 작업들

1) 온라인 데이터 저장소 스키마 수정

2) 신규 피쳐를 위한 데이터 동기화 파이프라인 개발

3) 새로운 피쳐들을 온라인 저장소로 backfill

4) 백엔드 서버에서 새로운 피쳐들을 사용하도록 하는 로직 추가/변경

문제

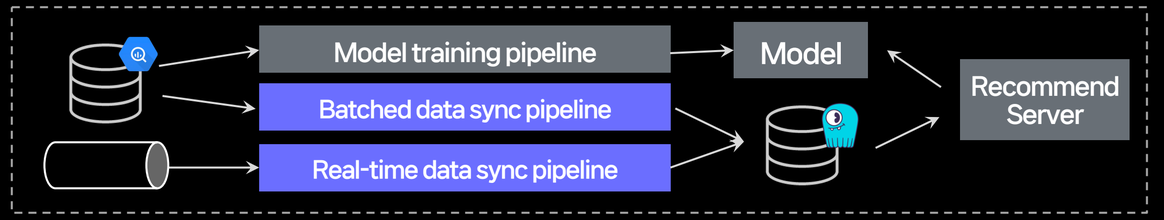
소프트웨어 개발 작업으로 인한 병목 때문에,
모델 온라인 실험의 iteration 속도가 늦어지는 현상 자주 발생

2.3 겪은 문제 3 - 다수의 추천 시스템 운영 시 컴포넌트 중복

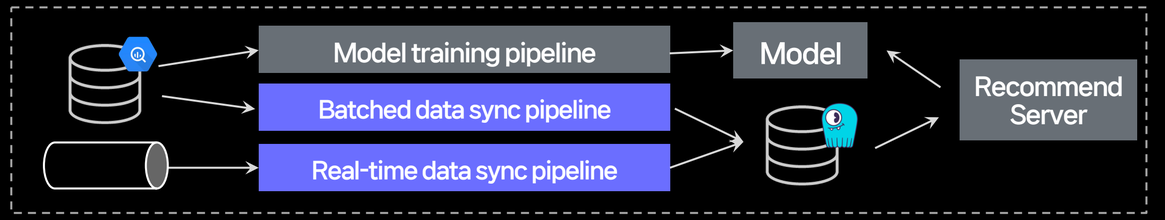
NAVER DEVIEW 2023

- 모델 학습 파이프라인은 추천 시스템마다 조금씩 다르며, 중복이 많이 발생하진 않았음
- 대신, 데이터 동기화를 위한 파이프라인은 많은 로직이 중복되고 있었음
  - ex) Offline DB connector, Online DB connector, Data validation logic, Parallel execution, Incremental update logic, Throughput limiter etc
- 이는 새로운 추천 시스템을 추가할 때마다 기술 부채로 작용

추천 시스템 1 (ex. Azar 1:1 Matching)

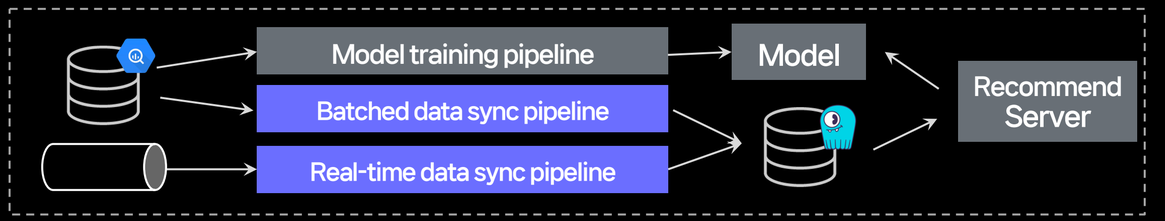


추천 시스템 2 (ex. Hakuna Live-room)



...

추천 시스템 N



2.3 겪은 문제 4 - 여러 추천 시스템 간 피쳐 공유의 어려움

NAVER DEVIEW 2023

- 하나의 서비스(ex. Azar)에서 여러 종류의 추천 시스템 (ex. 1:1 Matching, Live, Lounge) 이 있을 수 있음
- 유저 베이스가 같아도, 추천 시스템간 데이터 스키마와 온라인 저장소의 종류가 다를 수 있기에 피쳐 공유가 어려울 수 있음
  - 어떤 추천 시스템은 MongoDB를 온라인 저장소로 사용하면서, flatten key-value 형태의 자료구조를 사용하고, JSON을 이용한 직렬화를 할 수 있음
  - 다른 추천 시스템은 Redis를 온라인 저장소로 사용하면서, nested 형태의 자료구조를 사용하고, protobuf을 이용한 직렬화를 할 수 있음
  - 또 다른 추천 시스템은 Cassandra를 온라인 저장소로 사용하면서, DB에 직접 컬럼을 추가하여 직렬화를 할 수 있음
- 이는 여러 추천 시스템 간 피쳐 공유를 어렵게 만듦

2.4 Feature Store 도입 이유 & 역할

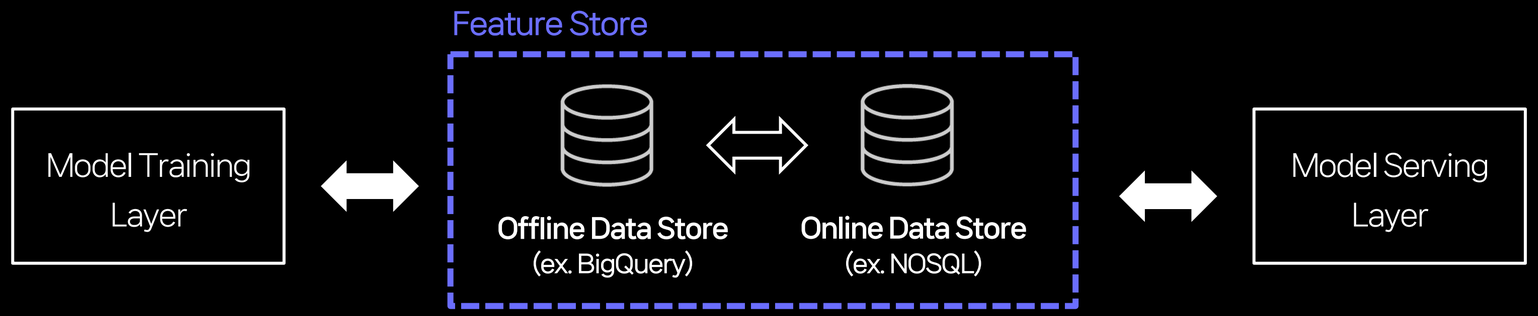
NAVER
DEVIEW
2023

Feature Store의 역할

- 학습/서빙 데이터 간 데이터 불일치 문제를 해결하고, 다수의 추천 시스템을 운영하며 필요한 다양한 컴포넌트를 하나로 모으는 일종의 플랫폼 역할 수행

도입 이유

- 다수의 추천 시스템을 운영하며 생겨난 여러 컴포넌트를 중앙화하며, 기술을 레버리지 하기 위해



Feature Store는 추천 시스템과 학습/서빙 데이터 저장소 사이의 일종의 abstracted layer이며,
추천 시스템이 더 본질적인 문제(추천)를 푸는데 집중할 수 있게 해줌

3. 하이퍼커넥트의 Feature Store

3.1 오픈소스? 자체 개발?

NAVER
DEVIEW
2023

하이퍼커넥트 추천 시스템의 요구사항

01 준 실시간으로 피쳐를 계산 및 사용할 수 있어야 함

- 기본적으로 정적인 아이템이 아닌 유저를 추천하는 경우가 많으므로, 실시간 피쳐가 성능에 큰 영향을 끼침
- 유저의 feedback 발생 후, 수 초 이내에 side-information 피쳐가 갱신될 수 있어야 함

02 Historical 피쳐에 대한 지원 필요

- Session-based recommendation 모델을 서빙 중이며, 따라서 historical 피쳐를 지원해야 함

03 오프라인 저장소로 BigQuery, 온라인 저장소로 ScyllaDB (Cassandra compatible) 지원 필요

- 사내에서 이미 메이저 하게 사용하는 기술 스택으로, 전사 인프라 관리 효율화 차원에서 기존 스택을 유지하는 것이 효율적

3.1 오픈소스? 자체 개발?

NAVER
DEVIEW
2023

자체 개발 결정

- 여러 오픈소스를 분석해 보았으나, 요구사항을 만족하는 오픈소스가 없어서 자체 개발 결정
- 가장 활성화된 오픈소스인 Feast와 In-house의 기능을 비교하면 아래와 같음

	Feast	In-house Feature Store
Historical feature 지원	X	O
Offline → Online 방향 데이터 동기화	O	O
Online → Offline 방향 데이터 동기화	X	O
온라인 저장소 실시간 업데이트 지원	△	O
Cassandra (ScyllaDB)를 온라인 저장소로 지원	X	O
Point-in-time Join 지원	O	X

† 2022년 5월경 PoC한 결과로, 현재는 다를 수 있습니다

3.1 오픈소스? 자체 개발?

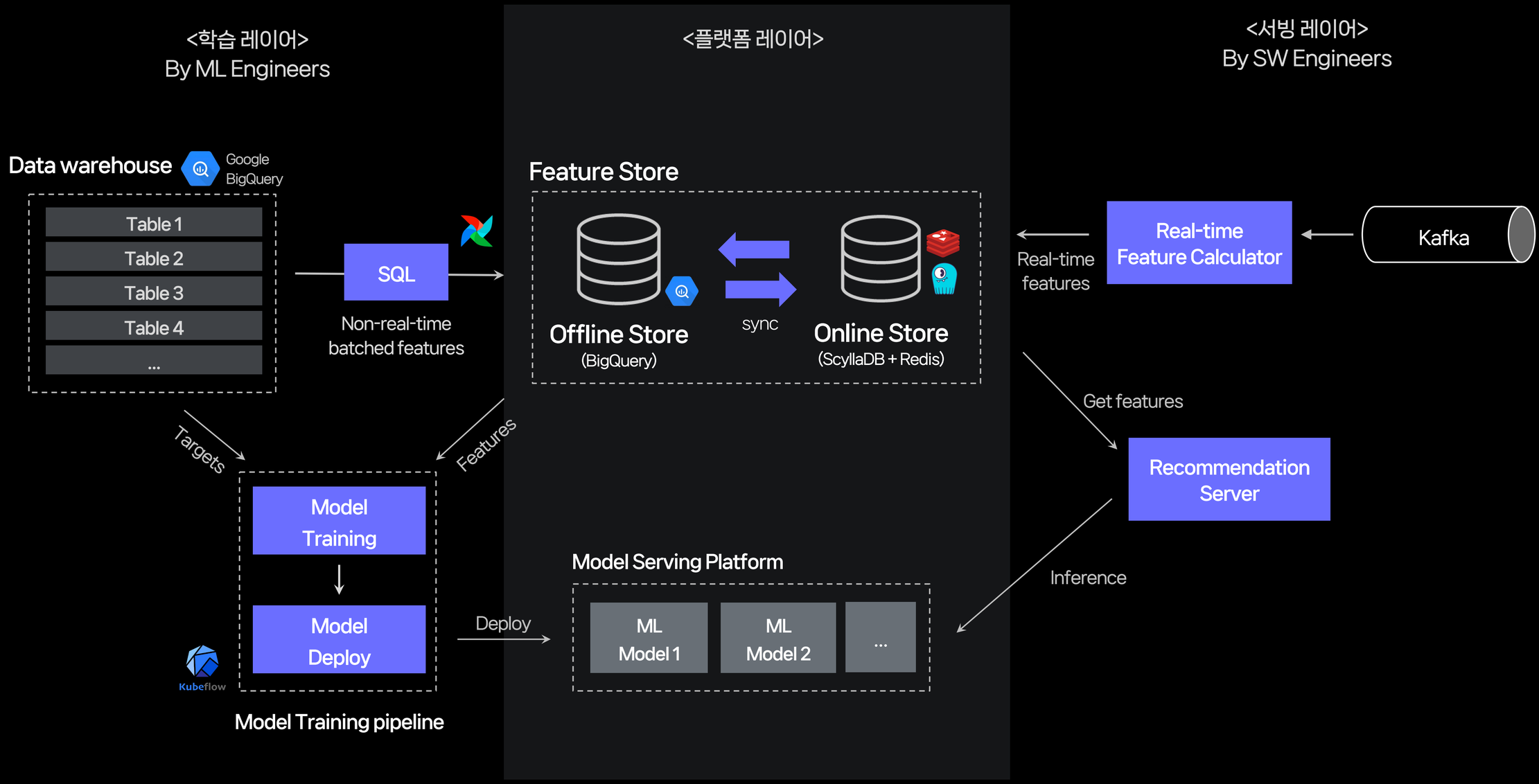
NAVER
DEVIEW
2023

In-house Feature Store의 Scope

- 자체 개발은 많은 리소스를 필요로 할 수 있기에, 가장 본질적인 문제를 해결하는 데에만 집중
- 추천 시스템을 위한 학습/서빙 통합 데이터 저장소를 만드는 문제만 우선 풀자!
- 부가적인 기능인 feature discovery, point-in-time join 같은 기능은 제공 X
  - Feature discovery (피쳐 탐색)은 이전처럼 BigQuery를 통해서 수행한다
  - Point-in-time Join은 기존처럼 SQL을 이용하거나, 스트리밍 어플리케이션에서 자체적으로 구현한다

3.2 Feature Store 이후의 하이퍼커넥트 추천 시스템

NAVER DEVIEW 2023



3.3 하이퍼커넥트 Feature Store의 기능들

NAVER
DEVIEW
2023

01 GitOps 기반의 피쳐 정의 시스템

02 Offline → Online 저장소 동기화 파이프라인 (Upsync)

03 Online → Offline 저장소 동기화 파이프라인 (Downsync)

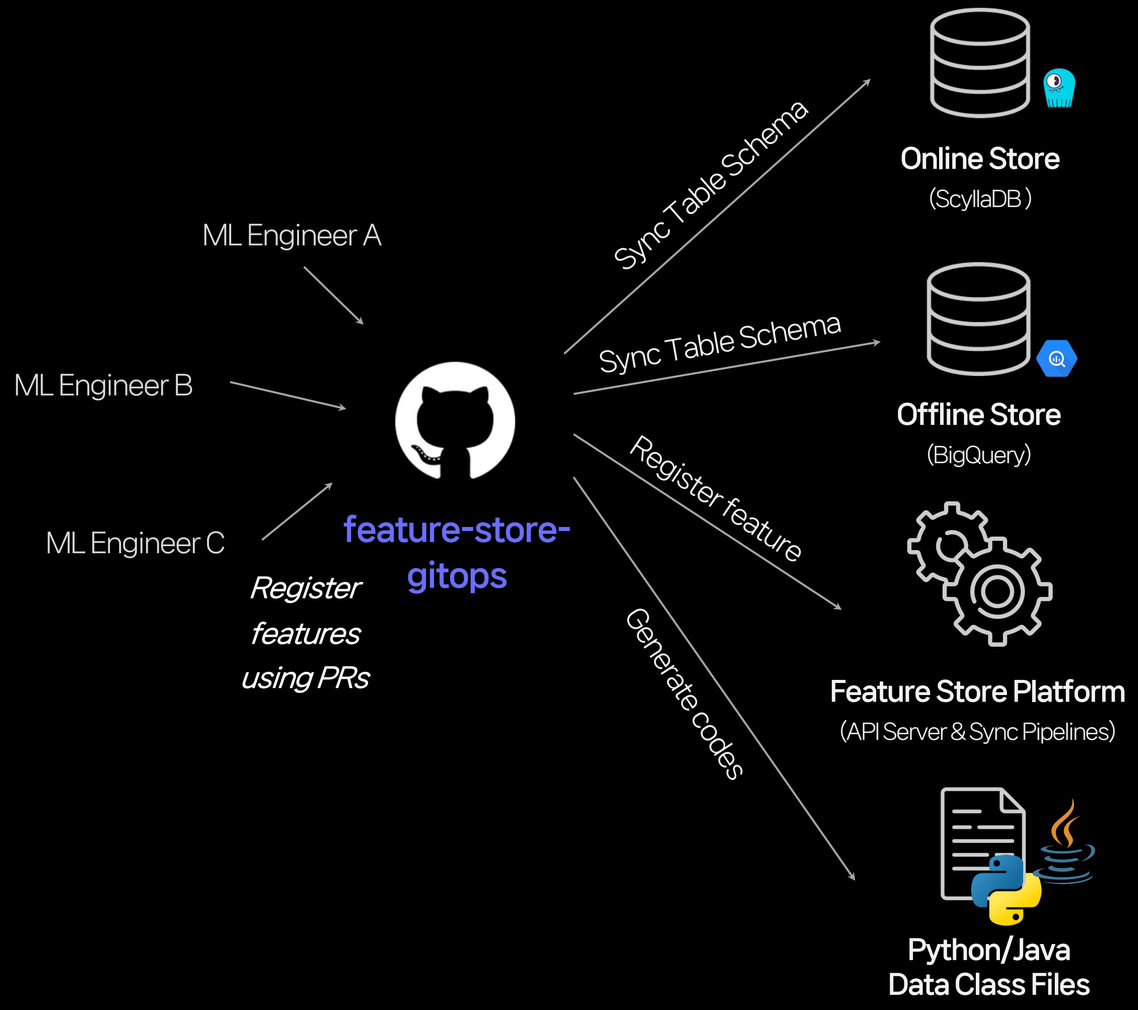
04 온라인 피쳐 Read API

05 접근제어 및 Data Governance 지원

3.3 기능 1 – GitOps 기반의 피쳐 정의 시스템

NAVER DEVIEW 2023

- Git으로 모든 피쳐에 대한 명세를 관리 중이며, **GitOps**를 이용하여 다양한 작업 자동화



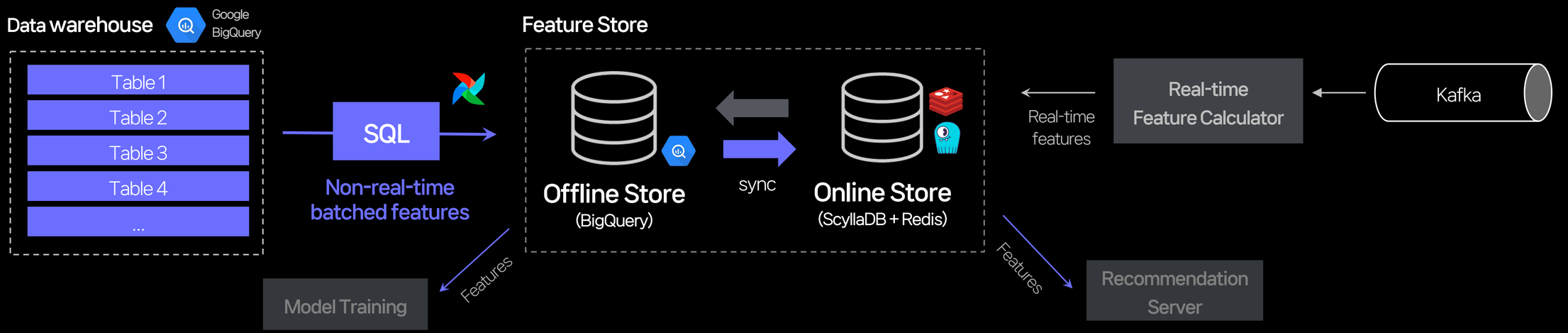

```yaml
live::hakuna-host-stat-v1:
  feature_type: Object
  owner: "shawn.s"
  schema:
    user_id:
      type: Text
      is_key_field: true
    gender:
      type: Text
      description: ""
      default_value: "MALE"
    level:
      type: Long
      description: ""
      default_value: 0
    country_code:
      type: Text
      description: ""
      default_value: "XX"
    birthday:
      type: Text
      description: ""
      default_value: "2000-01-01"
```

```yaml
azar-match::rich-history:
  feature_type: History
  owner: "ray.l"
  schema:
    user_id:
      type: Long
      description: ""
      is_key_field: true
    user_gender:
      type: Text
      description: ""
      default_value: ""
    user_country_code:
      type: Text
      description: ""
      default_value: ""
    user_language_code:
      type: Text
      description: ""
      default_value: ""
```

3.3 기능 2 – Upsync 파이프라인 (Offline → Online Store)

NAVER DEVIEW 2023

- 배치로 오프라인 저장소(BigQuery) 의 데이터를 **온라인** 저장소(ScyllaDB)에 동기화해주는 기능
- SQL 쿼리를 작성해 AirFlow에 파이프라인 등록만 하면 바로 오프라인 & 온라인 저장소에서 모두 사용 가능한 피쳐를 만들 수 있음
- SQL만 작성하면 끝이므로 사용하기 매우 편리하고 SW 엔지니어의 리소스도 불필요하지만, **실시간 피쳐는 사용할 수 없다는 한계 존재**



3.3 기능 2 – Upsync 파이프라인 (Offline → Online Store)

NAVER DEVIEW 2023

GitOps Yaml 등록 & AirFlow DAG만 작성하면 구성 완료!

GitOps Yaml

```yaml
azar-lounge::user-interaction:
  feature_type: Object
  owner: "shuki"
  schema:
    ...
  upsync:
    enabled: true
    incremental_update: true
```

* Incremental Update: 항상 모든 데이터를 동기화하는 것이 아닌, 마지막 동기화 시점 이후에 업데이트된 데이터만 해주도록 하는 옵션

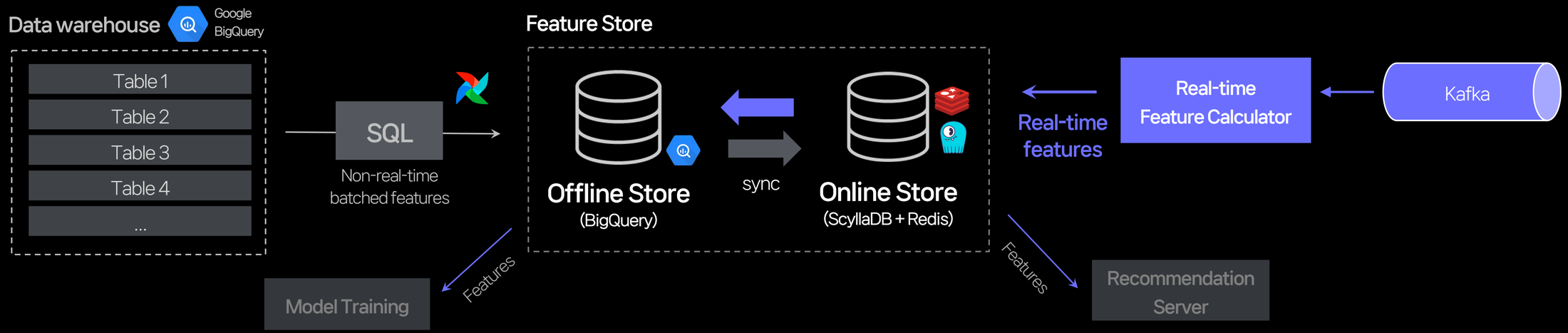
AirFlow DAG

```python
transform_user_stats = BigQueryOperator(
    task_id='transform_user_stats',
    use_legacy_sql=False,
    sql=open(f"{dirname}/query/transform/user_stats.sql").read(),
    dag=dag,
)

feature_store_upsync_worker_op = FeatureStoreUpsyncOperator(
    target_features=[
        "azar-lounge::user-interaction",
        "azar-lounge::user-stats",
    ],
)

[
    transform_user_interaction,
    transform_user_stats
] >> feature_store_upsync_worker_op >> end
```

3.3 기능 3 – Downsync 파이프라인 (Online -> Offline Store)

NAVER DEVIEW 2023

- 온라인(백엔드 서버)에서 실시간으로 피쳐를 계산해서 Feature Store에 등록하면, 오프라인 저장소(BigQuery)에도 동기화해주는 기능 (CDC와 비슷)
- SW 엔지니어 리소스를 필요로 하여 Upsync에 비하면 개발 공수가 조금 더 크지만, 실시간성이 중요한 모델이나 신규 가입자 케어가 필요한 상황에 유용함
- ex) 라이브 스트리밍 추천에서 실시간 시청자 수, 비전 피쳐, 신규 가입자의 click rate feature 등



3.3 기능 3 – Downsync 파이프라인 (Online -> Offline Store)

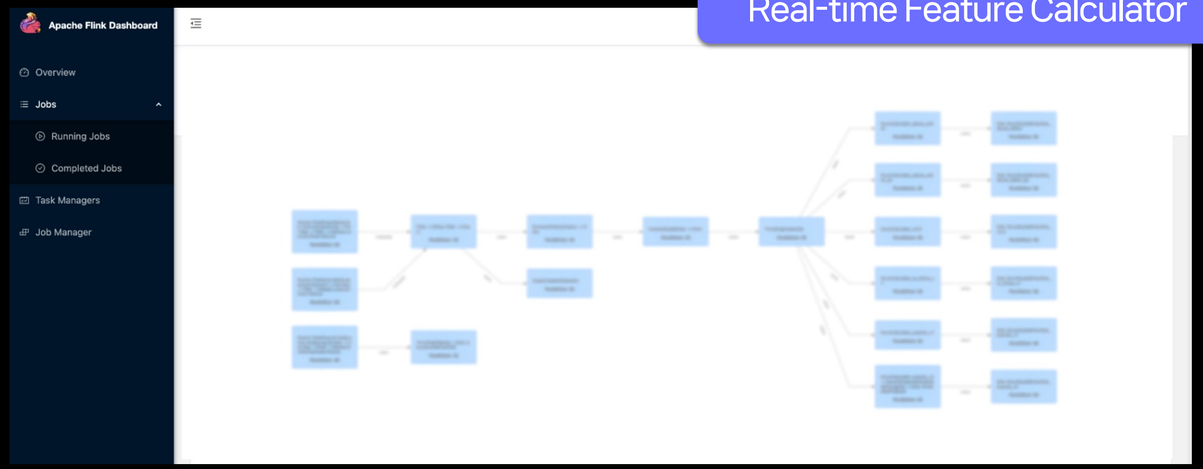
NAVER DEVIEW 2023

주로 Apache Flink와 같은 Event Streaming Application을 이용하여 실시간 피쳐 계산
피쳐 업데이트는 Kafka†에 Command를 보내는 방식으로 수행

GitOps Yaml

```
147  live::realtime-room-context:
148    feature_type: Object
149    owner: "owen.l"
150    schema:
         ...
183    downsync:
184      enabled: true
185      min_interval_by_key: "1m"
186      random_sampling_ratio: 0.9
```

Real-time Feature Calculator



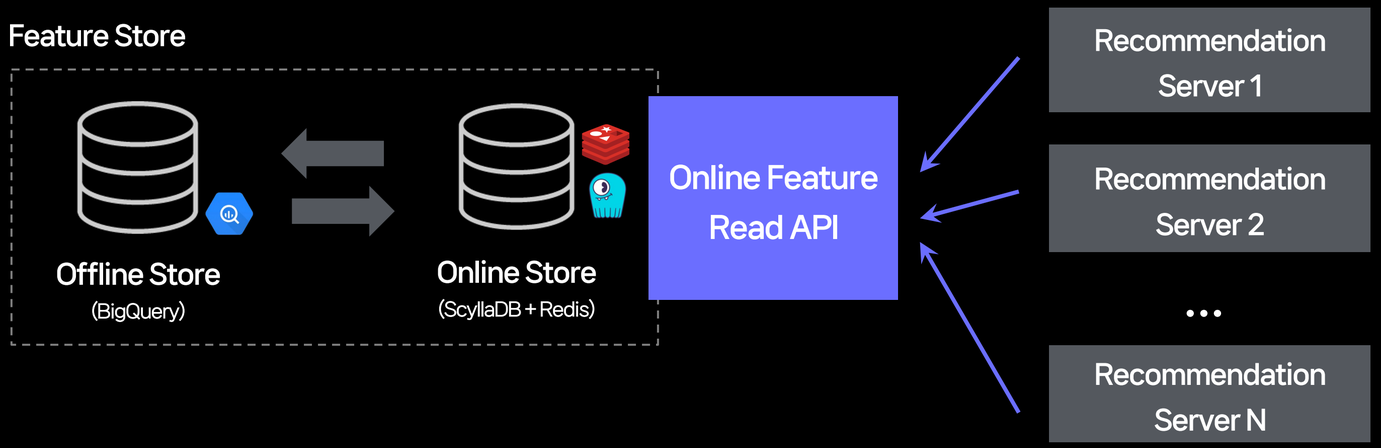
* Min Interval By Key 및 Random Sampling Ratio Update:
피쳐 업데이트 시 어떤 샘플링 정책으로 오프라인 저장소에 동기화할지에 대한 옵션

† 데이터 전송 및 스트리밍을 위한 오픈 소스 분산형 게시-구독 메시징 플랫폼

3.3 기능 4 – Online 피쳐 Read API

NAVER DEVIEW 2023

- 추천 서버들은 온라인 데이터 저장소에 직접 접근하는 대신 Read API를 통해 피쳐 접근
- API 서버를 통해 접근제어, 역직렬화(Avro), 캐시 옵션(Redis) 등을 지원
- 초창기 FastAPI로 개발해서 운영하다가, 현재는 성능 문제로 Spring으로 마이그레이션
- TPS: 수 천 이상 / p99 latency: 25ms†



† 99% request가 25ms 이하의 지연시간을 보이고 있음

3.3 기능 5 – 접근제어 및 Data Governance 지원

NAVER DEVIEW 2023

접근 제어

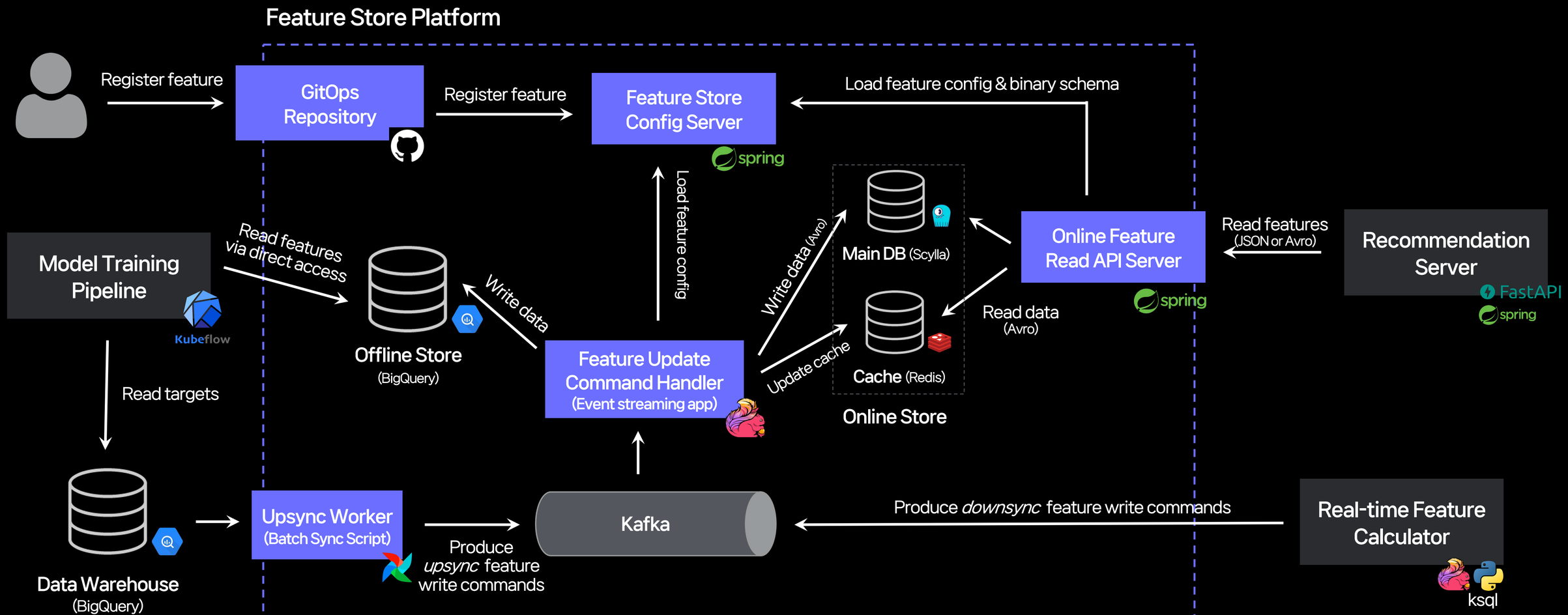
- 온라인 Read API에서는 마이크로서비스/개발자마다 접근 가능한 테이블을 설정 가능
- 오프라인 저장소에서는 BigQuery의 접근 제어 기능을 사용 중

Data Governance

- 오프라인 & 온라인 저장소 모두에서 데이터 리텐션을 통해 Data Governance 케어 중
- 비즈니스마다 리텐션 주기는 다르므로, 피쳐마다 주기를 설정할 수 있도록 제공 중

3.4 하이퍼커넥트 Feature Store 내부 아키텍처

NAVER DEVIEW 2023



3.5 하이퍼커넥트 Feature Store 사용법 요약

NAVER DEVIEW 2023

피쳐 정의 → GitOps에 등록

피쳐 쓰기 → 학습 레이어라면 SQL 작성 후 AirFlow를 통해서,
서빙 레이어라면 Kafka를 통해서

데이터 저장소간 동기화 → Offline → Online 방향이라면 Upsync 파이프라인 활성화,
Online → Offline 방향이라면 Downsync 파이프라인 활성화

활성화 방법: GitOps에 flag만 추가하면 끝!

피쳐 읽기 → 학습 레이어라면 BigQuery 사용,
서빙 레이어라면 Read API 서버를 통해서

4. 적용 사례 & 도입 효과

4.1 적용 사례

NAVER
DEVIEW
2023

Feature Store가 적용된 서비스

- 기존에 운영되던 대부분의 추천 시스템에 Feature Store 적용
  (Azar, Hakuna 서비스 내 5개 이상의 추천 시스템)
- 새롭게 시작한 추천 시스템들도 모두 Feature Store를 도입
- 추천 시스템 이외에도 이상 유저 탐지 시스템에도 Feature Store가 사용되고 있음

4.1 적용 사례

NAVER
DEVIEW
2023

Feature Store가 적용된 추천 시스템 유형

- Boosting 기반의 CTR(Click Through Rate) 예측 모델
- Deep learning 기반의 time spent 예측 모델
- 실시간 history 정보를 사용하는 세션 기반의 추천 시스템
- 실시간 비디오로부터 vision 정보를 추출하여 모델의 입력으로 사용하는 추천 시스템
  (라이브 스트리밍 한정)

4.2 도입 효과 > 데이터 일관성 문제 해결

NAVER
DEVIEW
2023

도입 이전

한 달에 한 번 꼴로 Data Inconsistency 문제 발견*

* BigQuery에서 분석한 피쳐 통계 값과, 실제 모델의 input으로 들어가고 있던 피쳐의 통계 사이에 큰 차이가 존재함을 발견

도입 이후

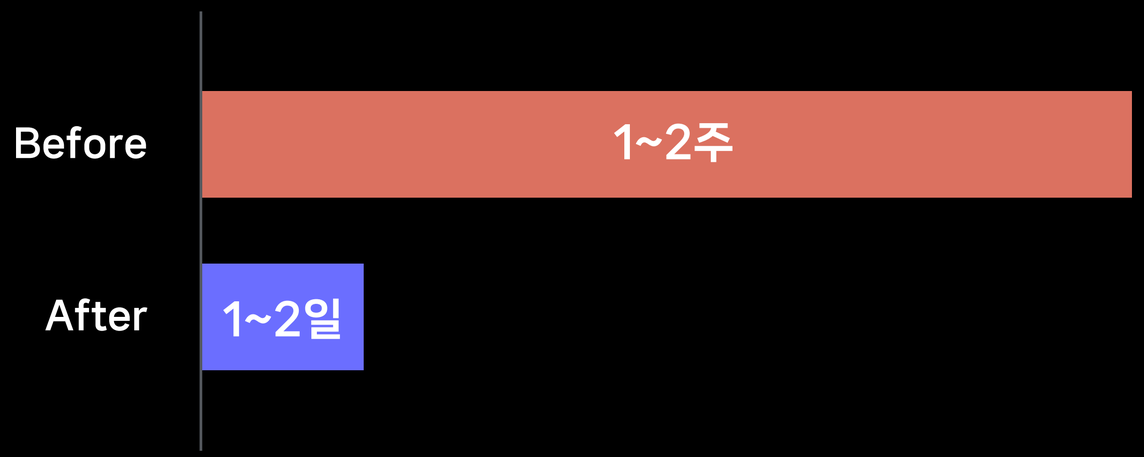
Feature Store 도입 이후 Data Inconsistency 문제 발견 X

4.3 도입 효과 > 개발 생산성

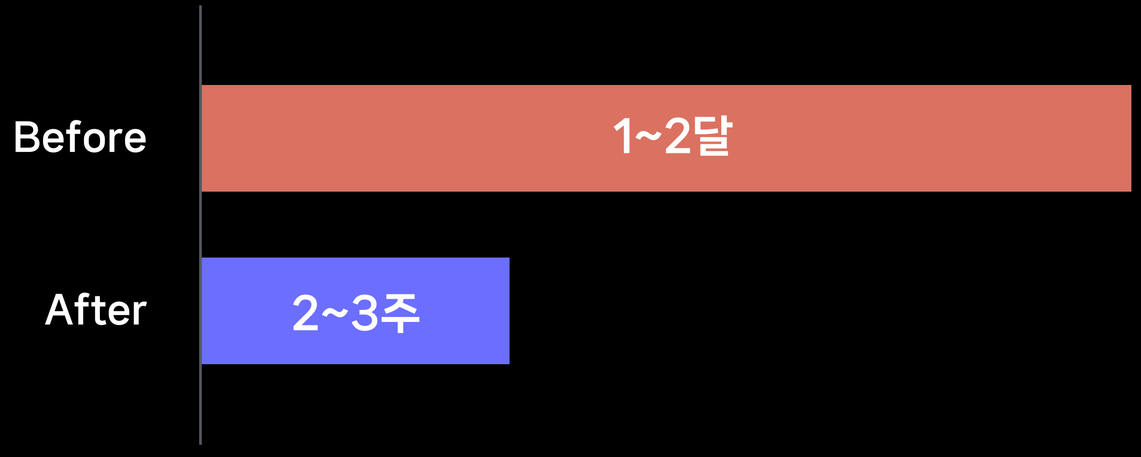
NAVER
DEVIEW
2023

서빙 레이어에서 신규 피쳐를 사용하기까지의 시간

(피쳐 엔지니어링 및 모델링 작업이 끝난 후, 신규 모델 실험까지 필요한 시간)



비실시간 피쳐만 사용하는 경우



실시간 피쳐까지 사용하는 경우

4.3 도입 효과 > 개발 생산성

NAVER
DEVIEW
2023

ML 엔지니어 (모델러) 입장에서의 효과

1. SW 엔지니어와의 커뮤니케이션 비용 감소
2. 한 번 만들어둔 피쳐를 여러 추천 모델에서 재사용 가능
3. 다른 ML 엔지니어가 만든 피쳐를 내가 만든 추천 모델에서 재사용~~(냘멱)~~ 가능

- ML 엔지니어: 모델에서 이런 피쳐를 사용하고 싶어요
- SW 엔지니어: 이 피쳐는 구조적으로 사용이 불가능합니다
- ML 엔지니어: 왜죠??

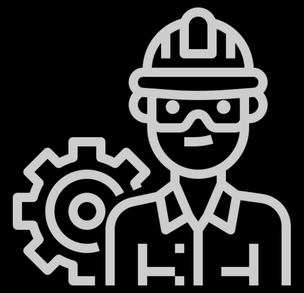
같은 대화 더 이상 NO!

4.3 도입 효과 > 개발 생산성

NAVER
DEVIEW
2023

SW 엔지니어 입장에서의 효과

1. 피쳐 동기화 파이프라인 및 로직 개발 시간을 아끼고, Core 로직 개발에 집중 가능
2. 학습/서빙 데이터 불일치 문제 신경 X



"이 피쳐 서빙 레이어에서 잘못 들어가고 있는 것 같은데 확인해주세요"
"제가 새롭게 추가한 피쳐 언제쯤 배포하고 실험 시작하나요?"

같은 말 더이상 듣지 않을 수 있음!

4.4 도입 효과 > 플랫폼화의 효과

NAVER
DEVIEW
2023

대부분의 추천 시스템이 Online Feature Read API를 사용하는 구조
→ 한 번의 기능 추가로 수많은 추천 시스템이 동시에 효과를 볼 수 있음

1. Redis Cache를 붙여서 동시에 추천 서버 latency 단축 효과를 봄
2. Binary 직렬화(Avro)를 도입하여 데이터 저장 비용을 아끼고, 네트워크 latency 단축 효과를 봄
   (Feature에 따라 최대 80% 이상 압축)
3. 이상 피쳐 탐지 시스템 도입 시 수많은 추천 시스템에서 효과를 볼 수 있음 (현재 개발 중)

4.5 도입 과정에서의 어려움

NAVER
DEVIEW
2023

- Feature Store를 실제로 적용하기 위해, 추천 시스템 여러 개에 Feature Store 연동 작업까지 직접 해줌
- 사내 Feature Store 설명회는 몇 번씩이고 반복 (ML 엔지니어, SW 엔지니어 따로)
- Online Read API 서버는 Python + FastAPI로 짰었는데, 성능(latency + throughput)이 너무 안 나와서 Kotlin + Spring으로 다시 짜기도 함
- 사내 공용 분산 DB(Scylla)의 가장 큰 고객 중 하나가 되어버려서, DevOps 팀과의 상담(?)도 자주 함

Q & A

Thank You