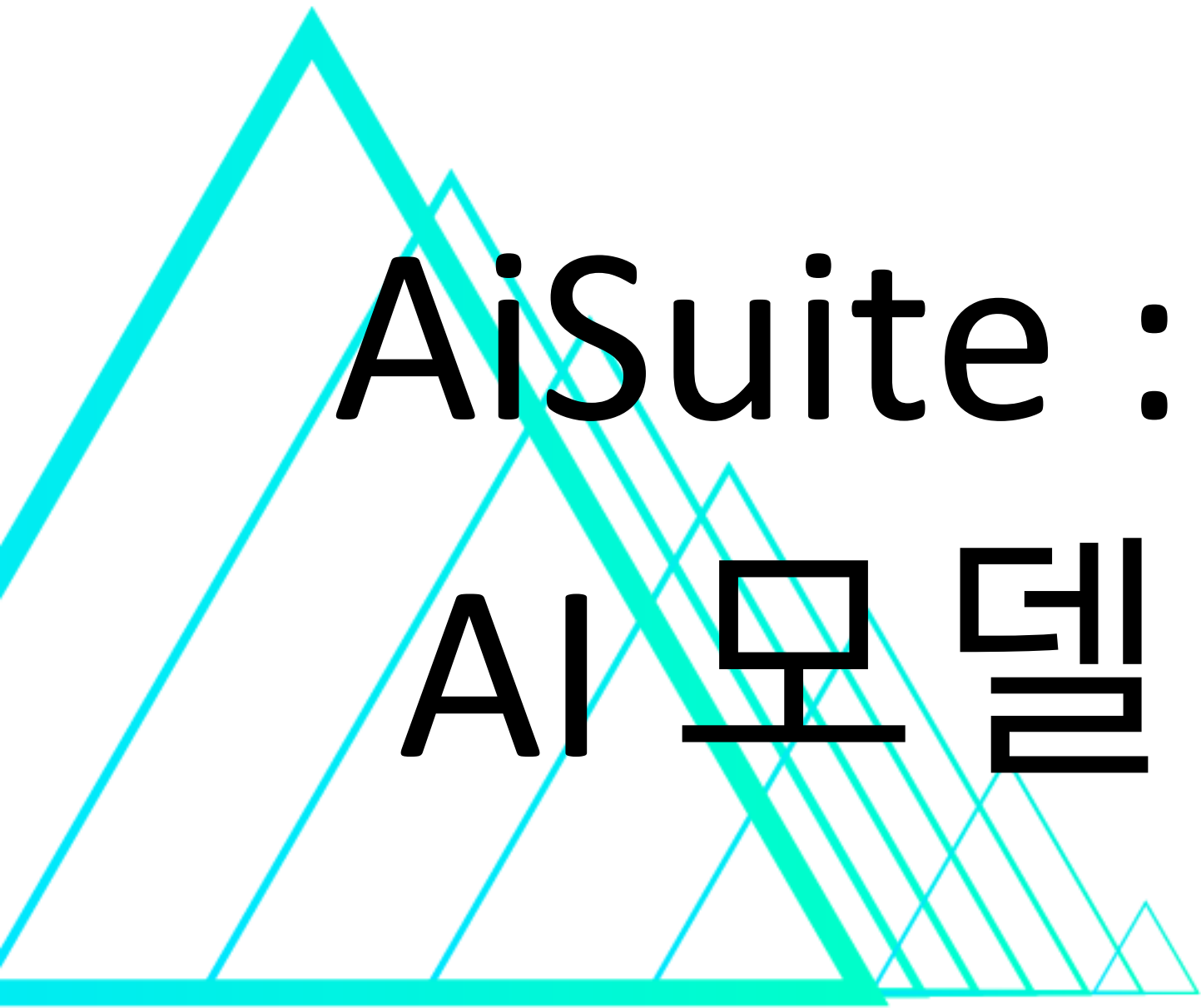




AiSuite : Kubeflow를 통해 더 나은 AI 모델 서빙과 MLOps 실현하기



한완규 NAVER
박정욱 NAVER

CONTENTS



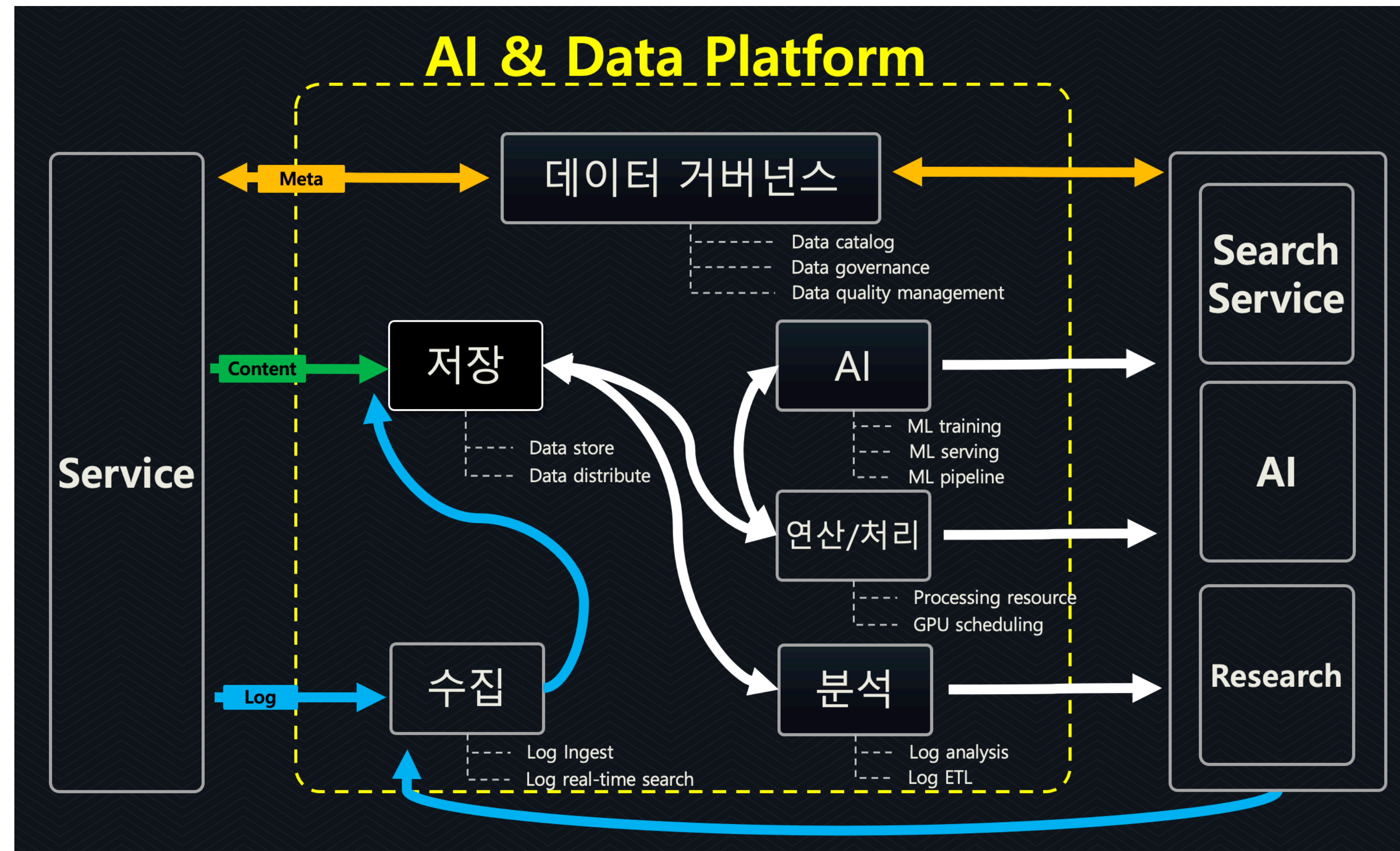
1. 왜 Kubeflow 기반의 AI 플랫폼인가?
 2. KFServing으로 대규모 ML 서비스 서빙하기
 3. Kubeflow로 MLOps 실현하기
 4. Multi-Tenant Kubeflow
 5. HDFS를 k8s 볼륨으로 사용하기 위한 Alluxio 연동
 6. Future Work
- 

1. 왜 Kubeflow 기반의 AI 플랫폼인가?

1.1 Hadoop 기반 사내 AI 플랫폼

네이버는 데이터 센터를 직접 운영하고 있고, 저희 AI & Data Platform 부서에서는 그 위에서 **Hadoop 기반 사내 빅데이터 플랫폼**을 제공합니다.

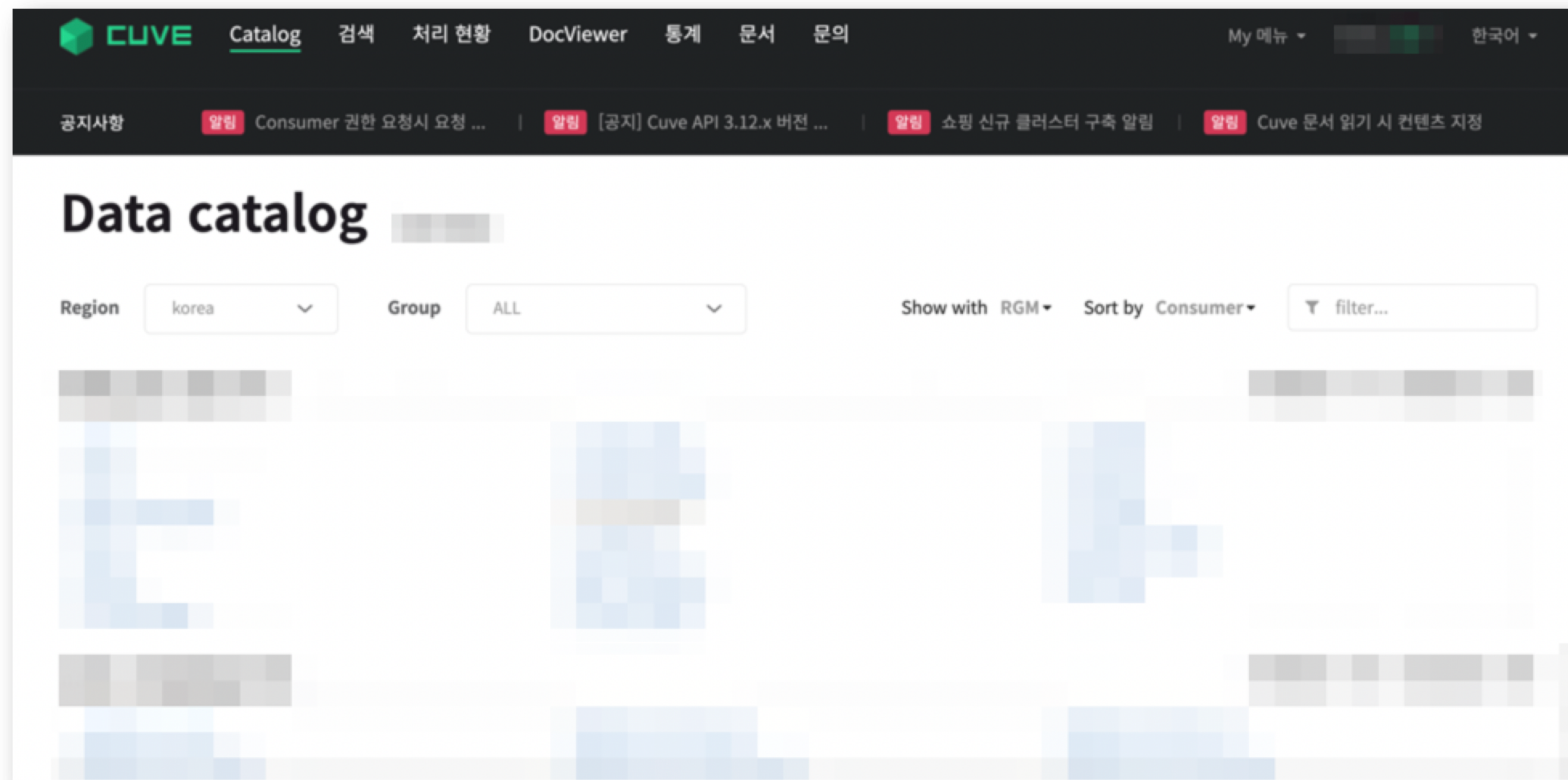
- 네이버 검색, 네이버 쇼핑 등 전사의 다양한 서비스들이 저희 플랫폼을 이용하고 있습니다!



1.1 Hadoop 기반 사내 AI 플랫폼

Cuve : 사내 분산 저장 플랫폼. 수많은 네이버 서비스들의 로그가 저장됨

CQuery : Cuve 저장소의 로그를 SQL로 조회하기 위한 프레임워크



```
from cquery import hive

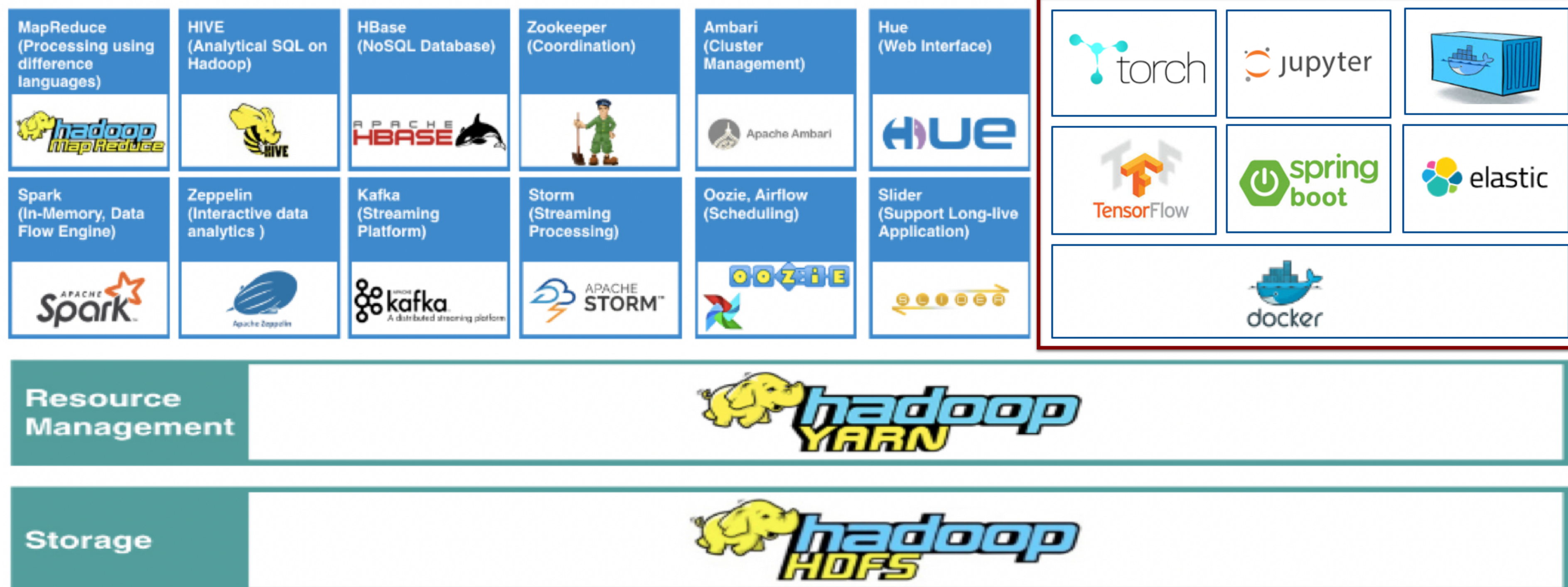
# 1. connect lago hive using keytab file
conn = hive.connect(location=...,
                    principal=...,
                    ticket=...,
                    keytab_file=...)

cursor = conn.cursor()
try:
    cursor.execute(f"""
select count(*) from ...
where log_date='2021-09-01'
""")
    cnt = cursor.fetchone()[0]
finally:
    # 2. close connection
    cursor.close()
    conn.close()
```


1.1 Hadoop 기반 사내 AI 플랫폼

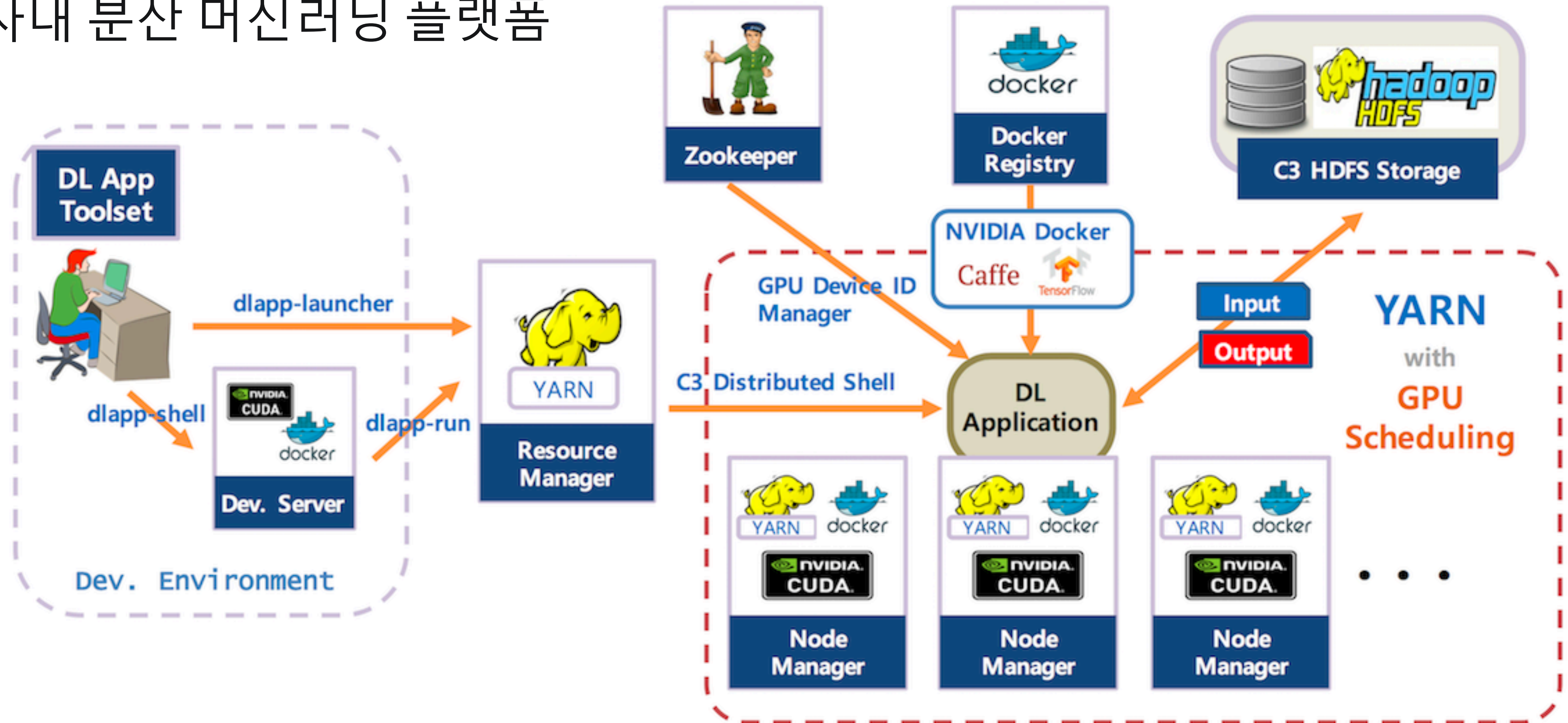
1.4 C3 현재

DEVIEW
2018

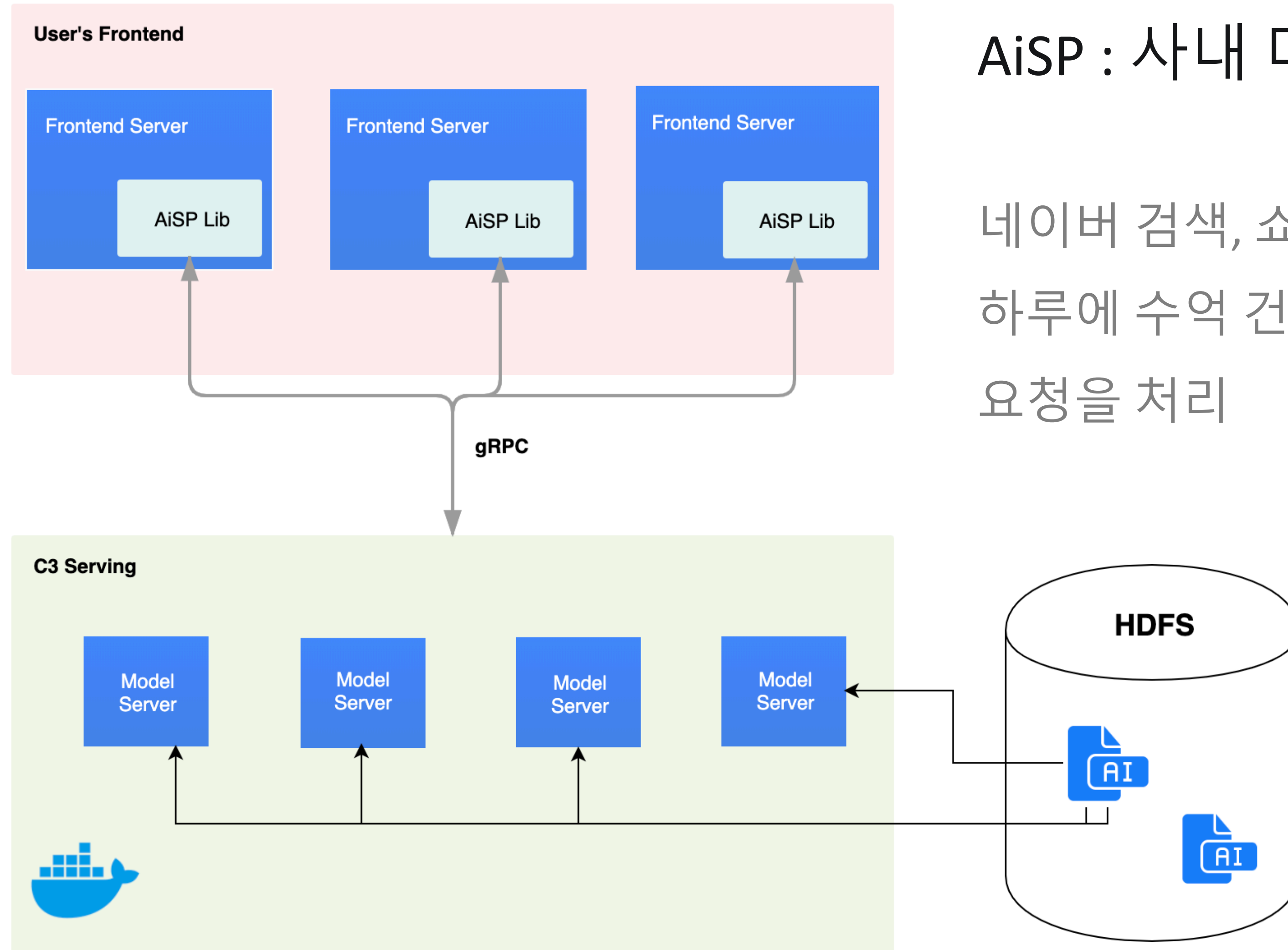


1.1 Hadoop 기반 사내 AI 플랫폼

C3DL : 사내 분산 머신러닝 플랫폼



1.1 Hadoop 기반 사내 AI 플랫폼



AiSP : 사내 머신러닝 서빙 플랫폼

네이버 검색, 쇼핑 등 여러 서비스에서 하루에 수억 건의 머신러닝 인퍼런스 요청을 처리

1.1 Hadoop 기반 사내 AI 플랫폼

쇼핑, 웹, 이미지, 스마트어라운드, 클로버 OCR 등의 서비스

C3DL

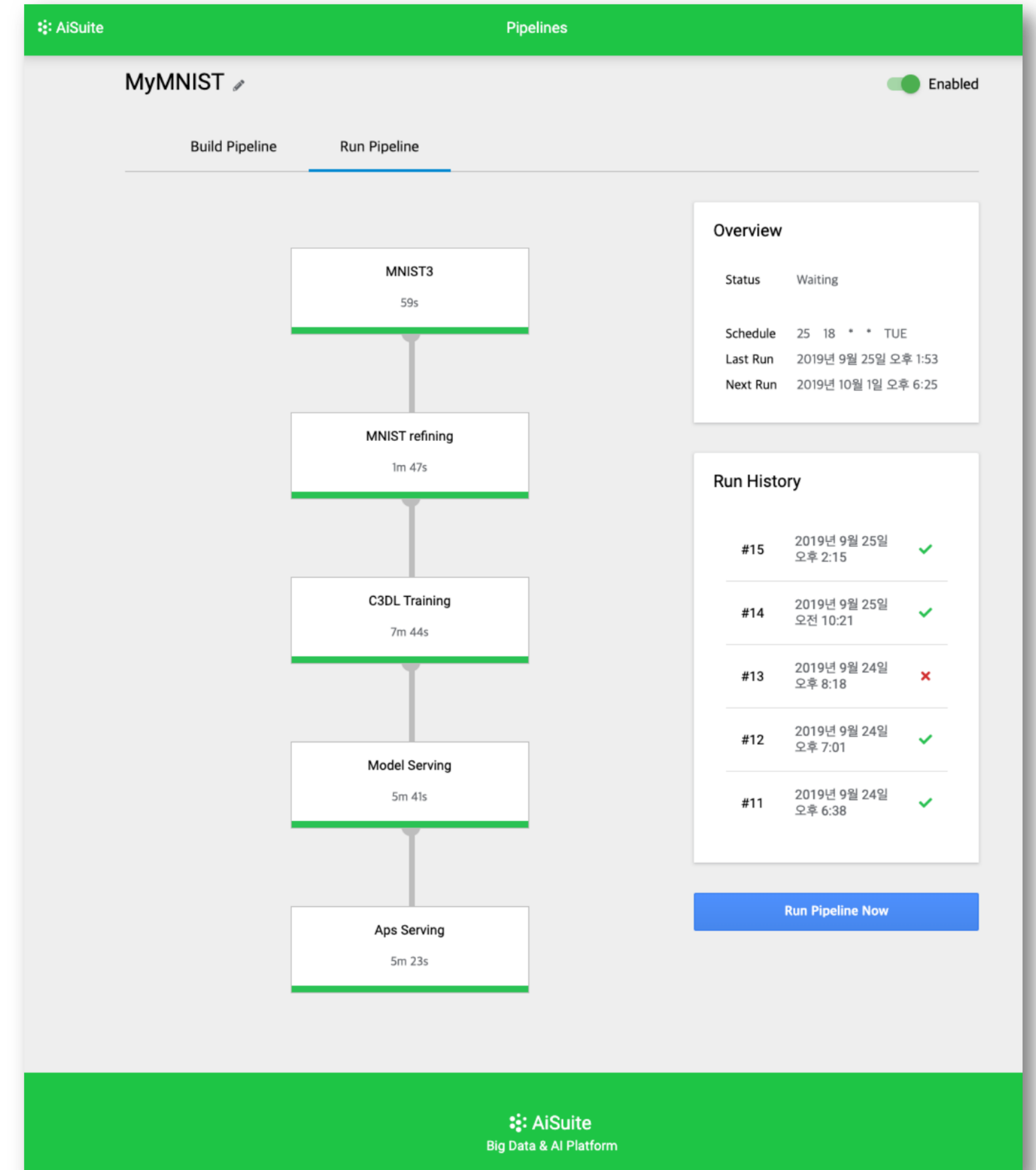
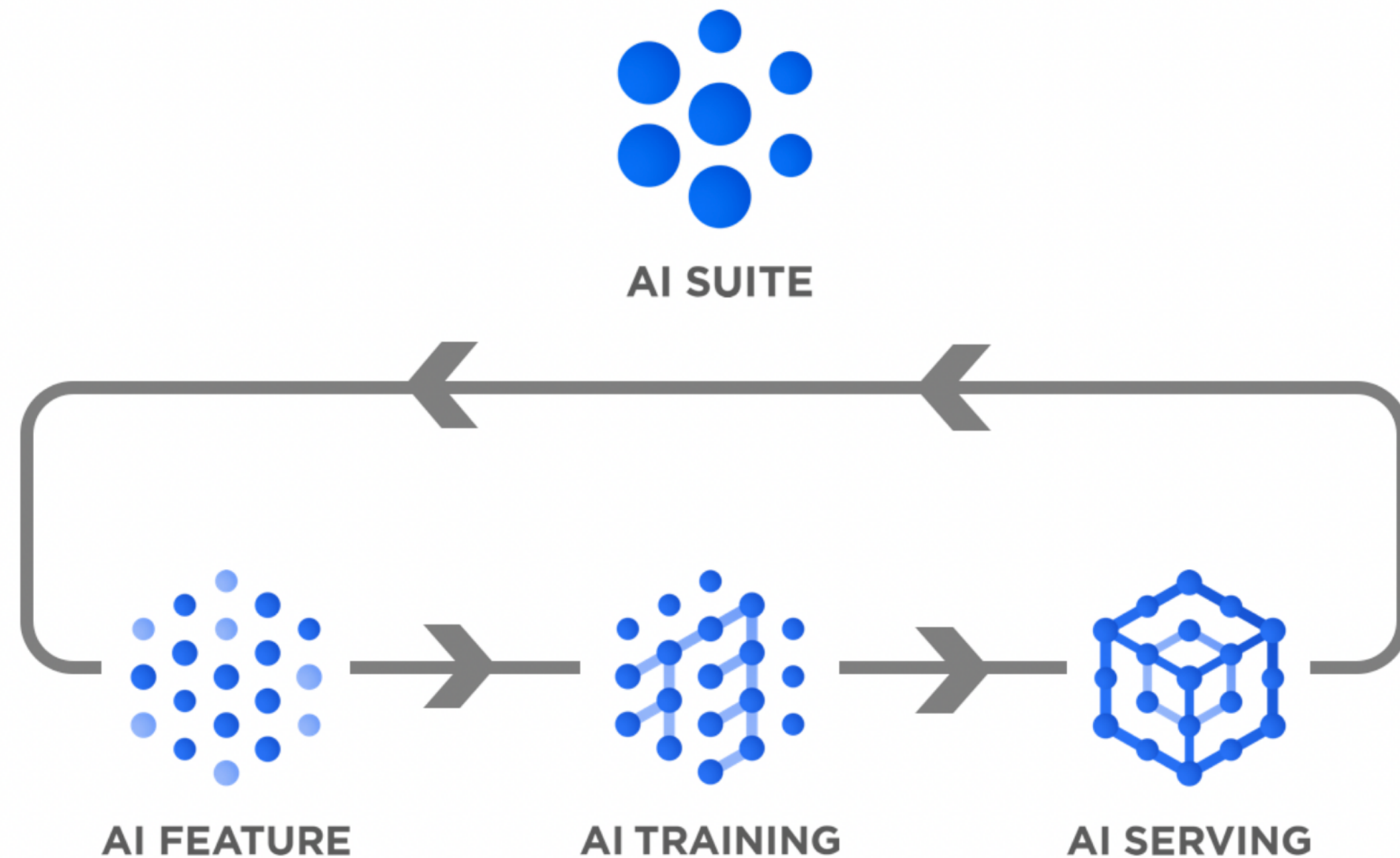
AiSP

Hadoop YARN / Slider (C3S)

Hadoop HDFS

1.1 Hadoop 기반 사내 AI 플랫폼

사내 머신러닝 파이프라인 플랫폼



1.2 한계점

Hadoop 은 “빅데이터 처리”를 하기 위해 만들어진 플랫폼

컨테이너 오케스트레이션의 제약

- 배포, 스케일링, 모니터링 등 컨테이너 관리 기능의 한계

컨테이너 네트워킹을 위한 추가 개발 필요

- 오버레이 네트워크 지원을 위해 Flannel 및 DNS 등을 직접 커스터마이징 해야함

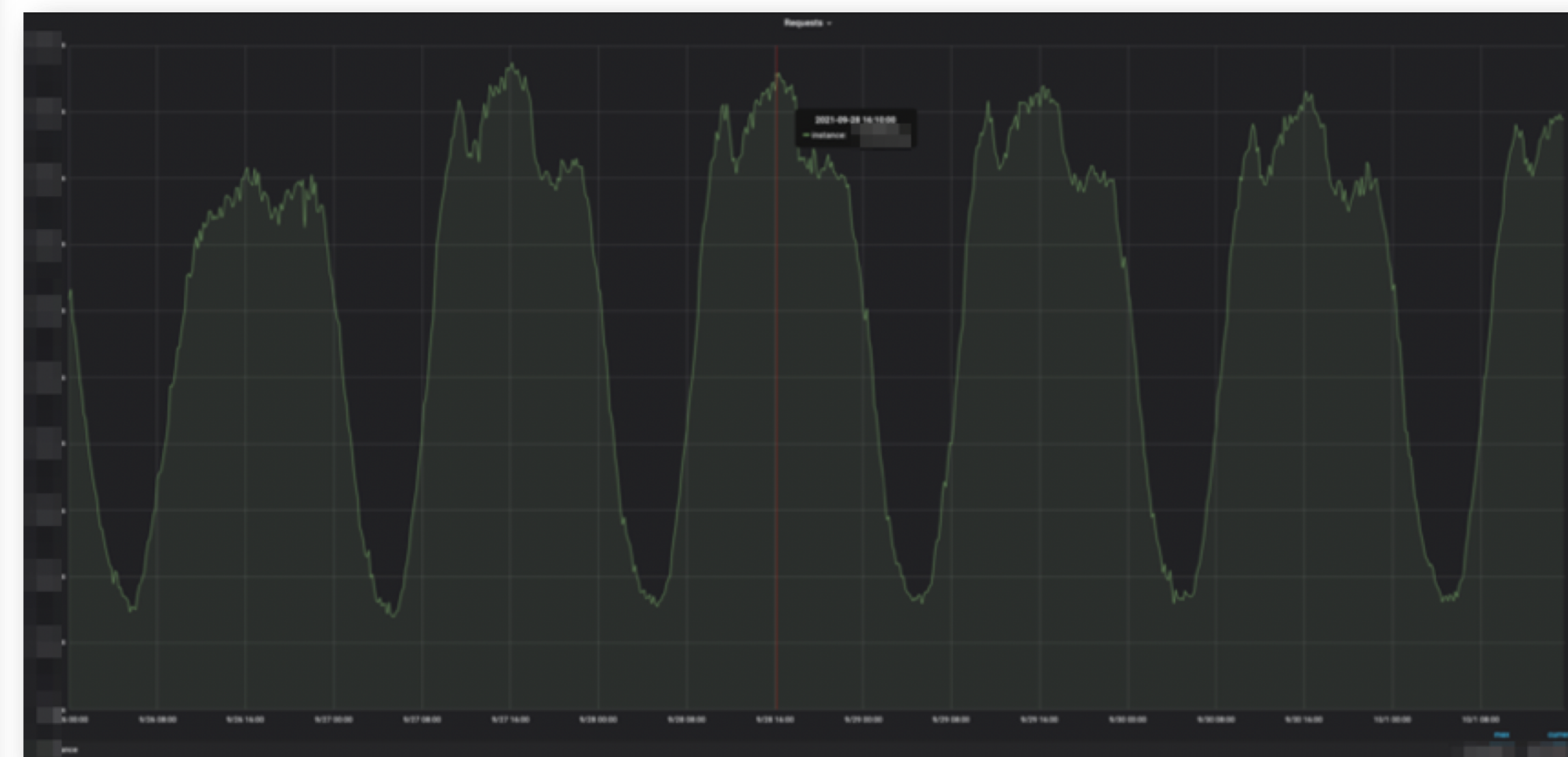
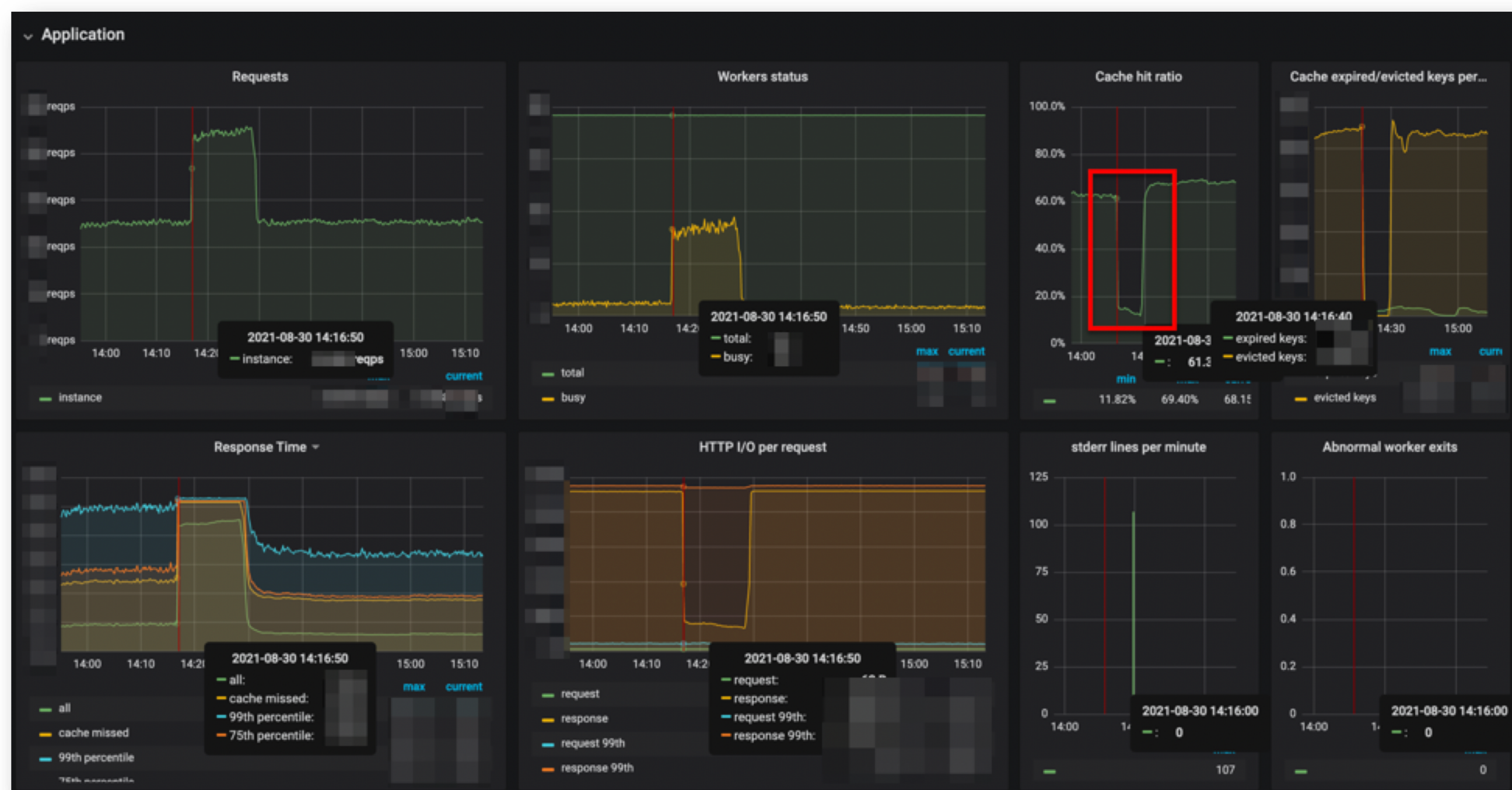
GPU 스케줄링의 한계

- Hadoop 내 GPU 지원의 한계로, GPU 스케줄러를 직접 개발해서 사용함

1.2 한계점

서빙 플랫폼 운영시 겪은 어려운 점들

- 블루/그린 배포로 인한 한계점 (급격한 캐시 성능 저하, 리소스 2배 필요)
- 오토스케일러 x. 요청이 급격하게 증가했을 때 수동으로 대응 필요

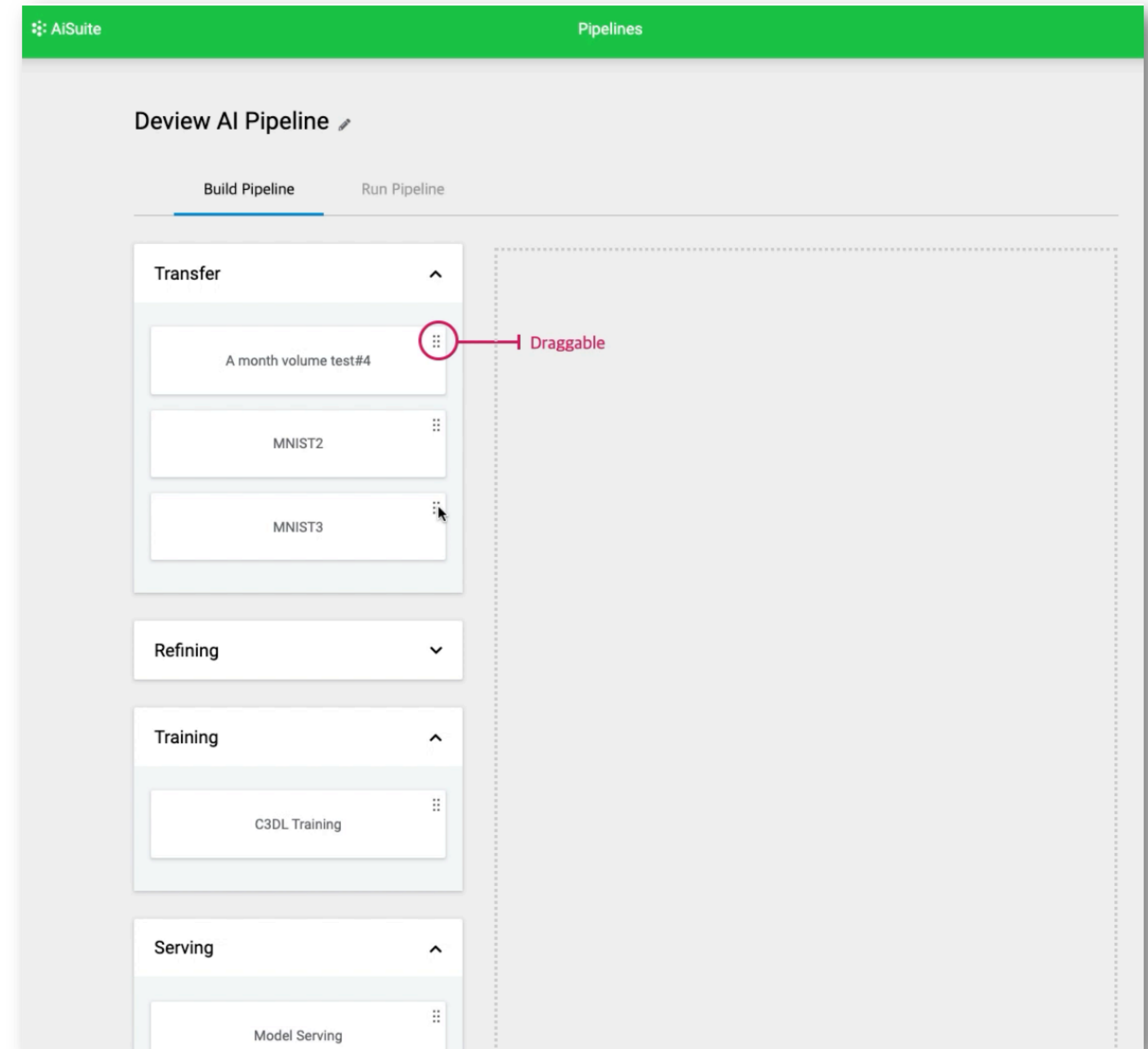


1.2 한계점

ML 파이프라인 제공의 어려움

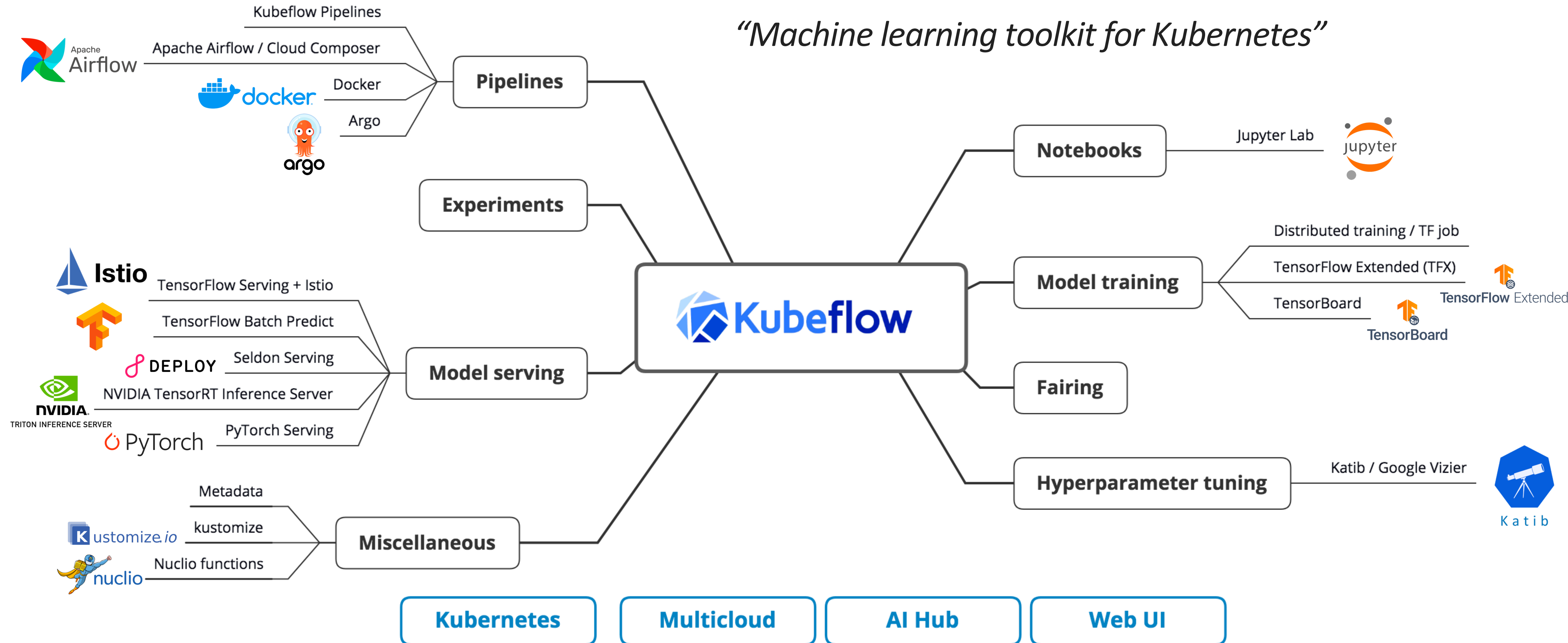
- 인프라 관리팀이 아닌 사용자가 직접 모듈을 추가하기 어려움 (모듈 로직 외 프론트엔드 로직 등의 추가 개발이 필요)

aps_template.html	Allow user to write multiple aps config files (#222)
base.html	Update cache numbers
build_pipeline.html	Show message to create job if none to list
dump_template.html	Show hdfs namespace
home.html	Home polish (#133)
inference_template.html	Let user set japan employee number
list.html	Use real enum type for status (#214)
pipeline.html	update hash date
refine_template.html	Implement run pipeline page (#70)
run_pipeline.html	More intuitive dates (#120)
training_c3dl_template.html	Add tooltips (#115)



1.3 Kubeflow 관?

“Machine learning toolkit for Kubernetes”



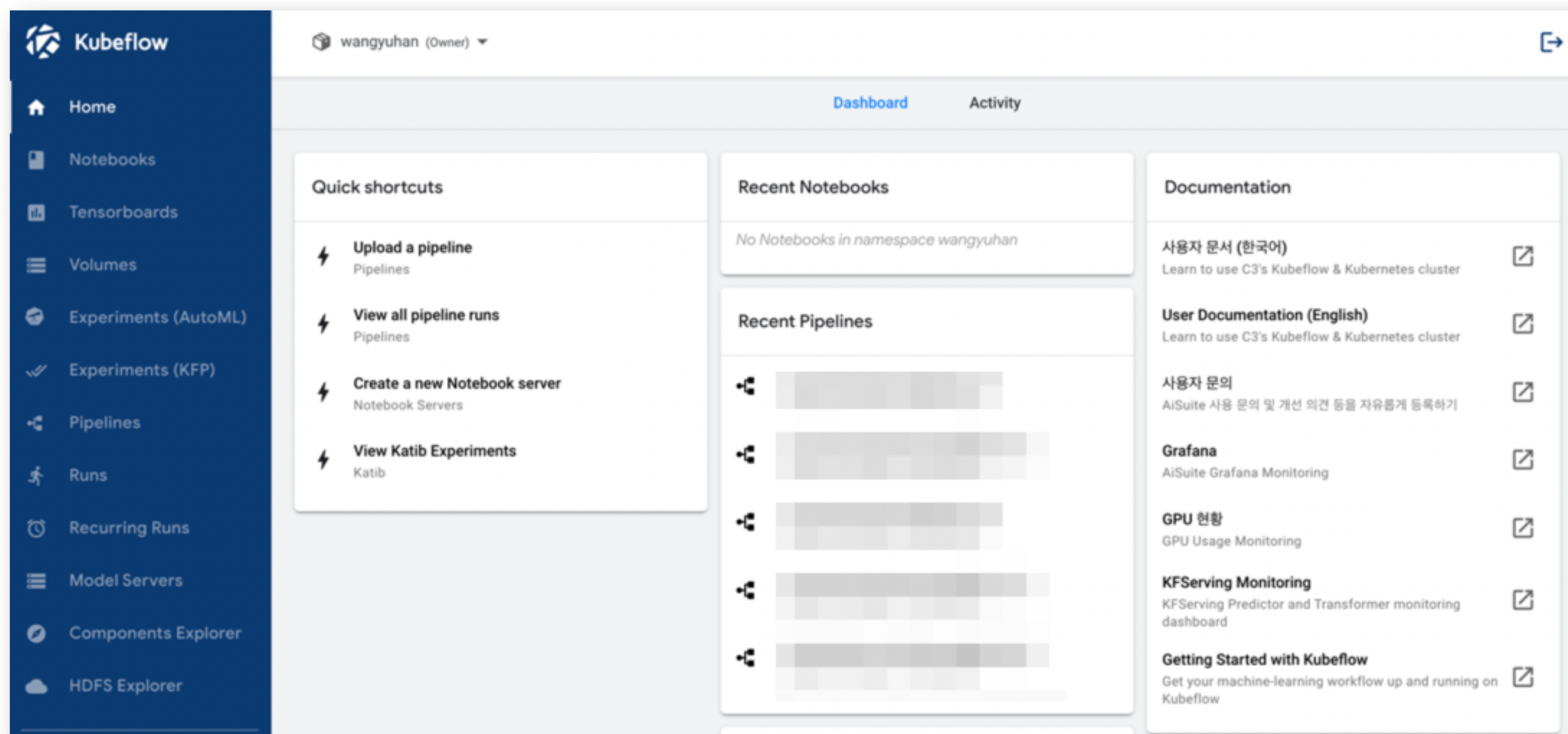
1.4 Kubeflow 기반 AI 플랫폼의 장점

다양한 ML 프레임워크 지원

- Tensorflow, PyTorch, ONNX, scikit-learn, MXNet, XGBoost, Chainer, NVIDIA Triton Inference Server

Kubeflow 대쉬보드 제공

- 사용자 문서를 추가하거나,
새로운 탭을 추가하는 등
커스터마이징도 손쉽게 가능



1.4 KubeFlow 기반 AI 플랫폼의 장점

k8s 컨테이너 오케스트레이션

- CNI 플러그인을 통한 네트워크 구성 제공
- Prometheus 호환을 통한 손쉬운 모니터링 구성 제공
- 라이브니스, 레디니스 프로브, 사이드카 컨테이너 기능
- Istio를 통한 카나리 배포 지원
- Knative를 통한 오토스케일링, 제로 스케일, 서버리스 지원

1.4 Kubeflow 기반 AI 플랫폼의 장점

GPU 지원 용이

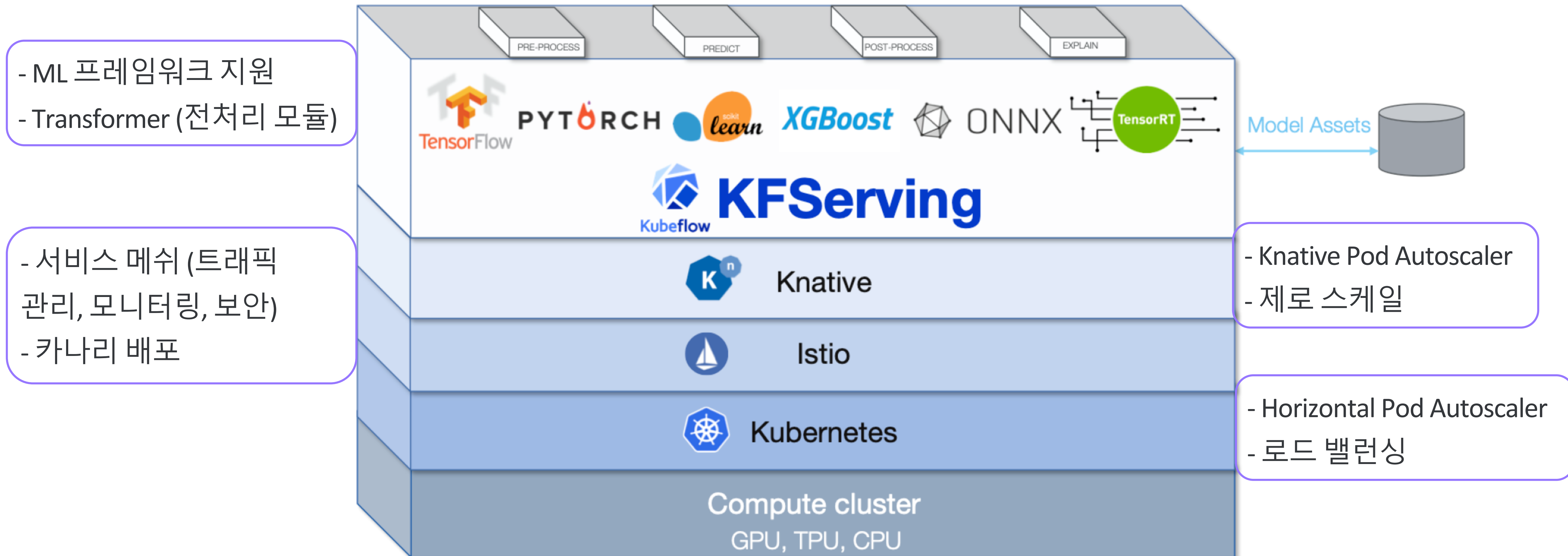
- Nvidia, AMD GPU 디바이스 플러그인 지원
- 노드 레이블과 노드 셀렉터를 이용해 GPU 종류별 스케줄링 가능

다양한 k8s 에코시스템

- Istio, Knative, Argo CD, Deepops, Helm, Kubespray 등등...
- k8s에서 유용하게 사용할 수 있는 새로운 프레임워크들이 활발하게 개발되고, 공유되고있음

2. KFServing으로 대규모 머신러닝 서비스 서빙하기

2.1 KFServing 이란?



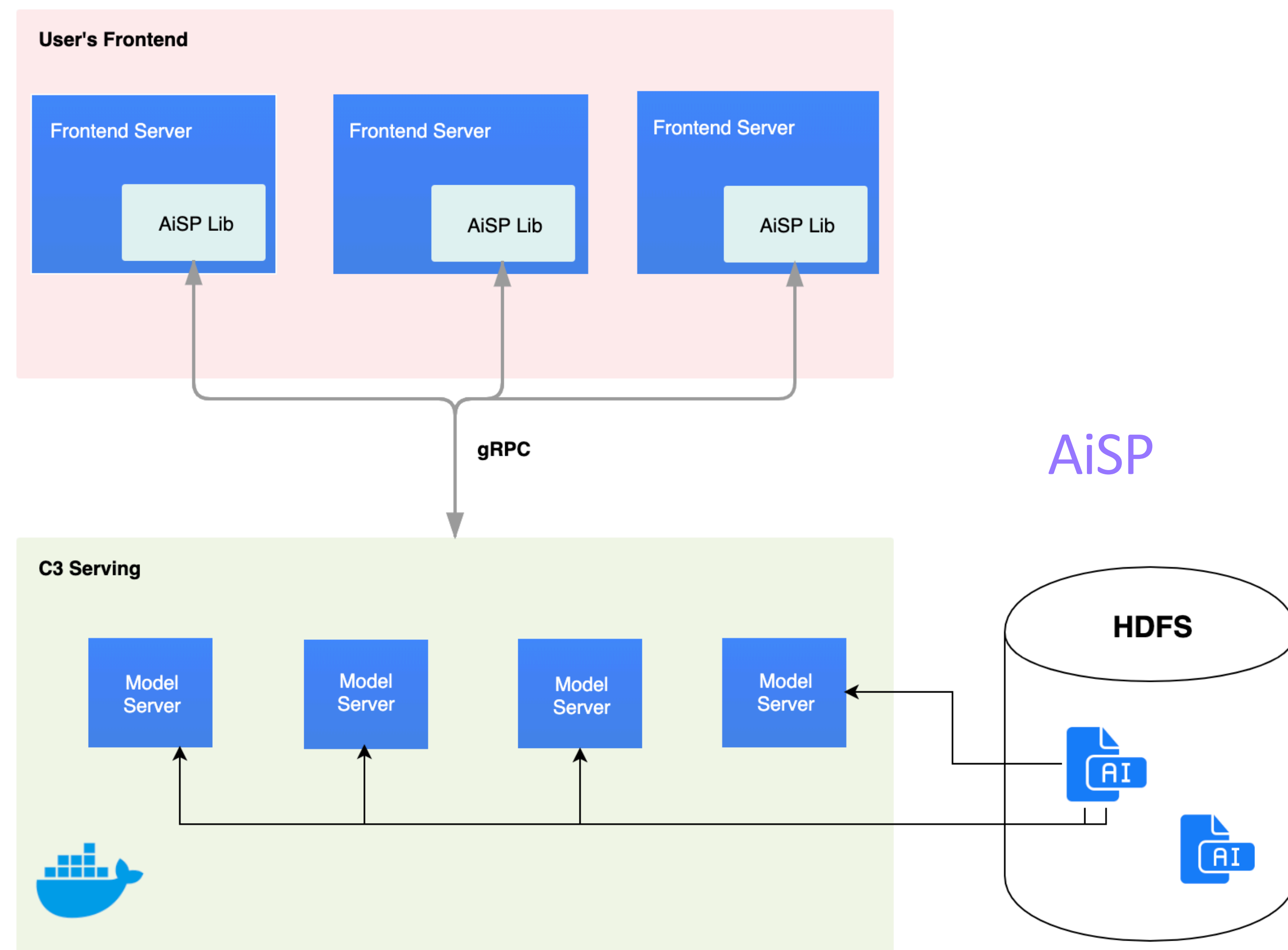
2.2 기존 서빙 플랫폼과의 비교

C++, Python 기반 클라이언트
라이브러리

- 서비스 디스커버리, 로드 밸런싱,
데이터 직렬화, 모니터링, 캐싱

Tensorflow, ONNX, LightGBM 기반
모델서버

- C3S HDFS에서 모델을 로딩
- gRPC 통신

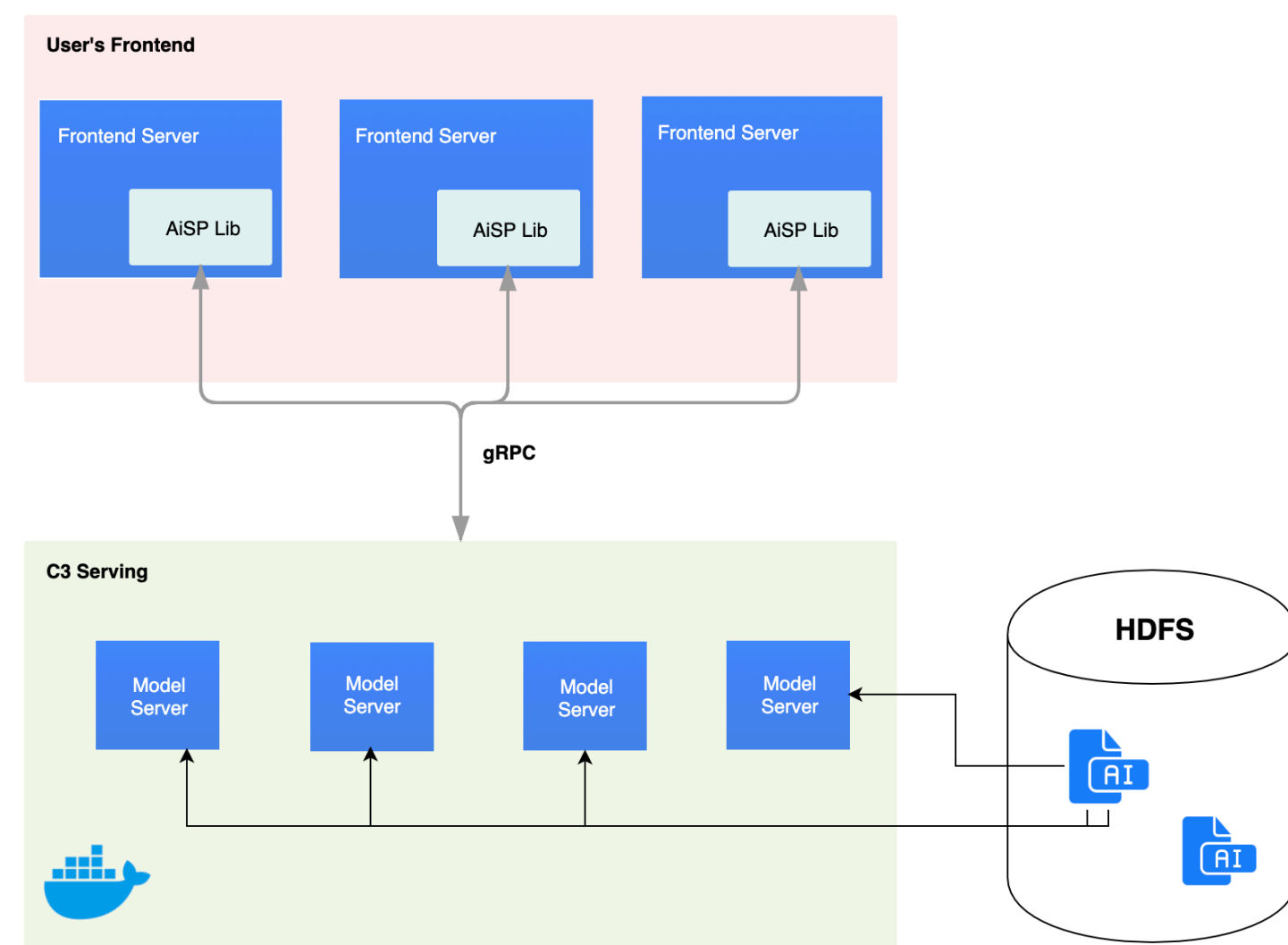


2.2 기존 서빙 플랫폼과의 비교

사용 방식

클라이언트 사이드 로드밸런싱

=> 부하 분산이 완전히 균등하지 않고,
서버 추가 시 분산에 시간이 걸리는 문제
프로그래밍 언어 제약 (C++, Python)



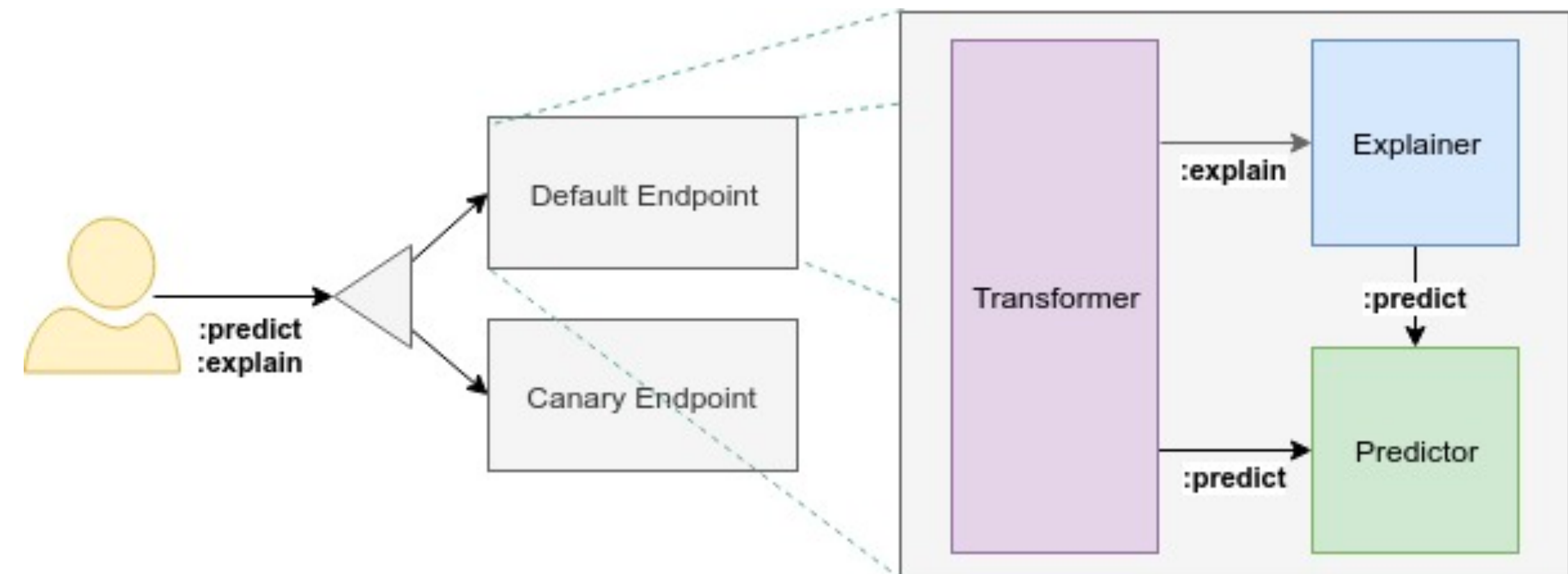
AiSP

KFServing

서버 사이드 로드밸런싱

=> 균등한 부하 분산

gRPC, REST API를 통한 단일 진입점 제공
(어떤 프로그래밍 언어든 사용 가능)



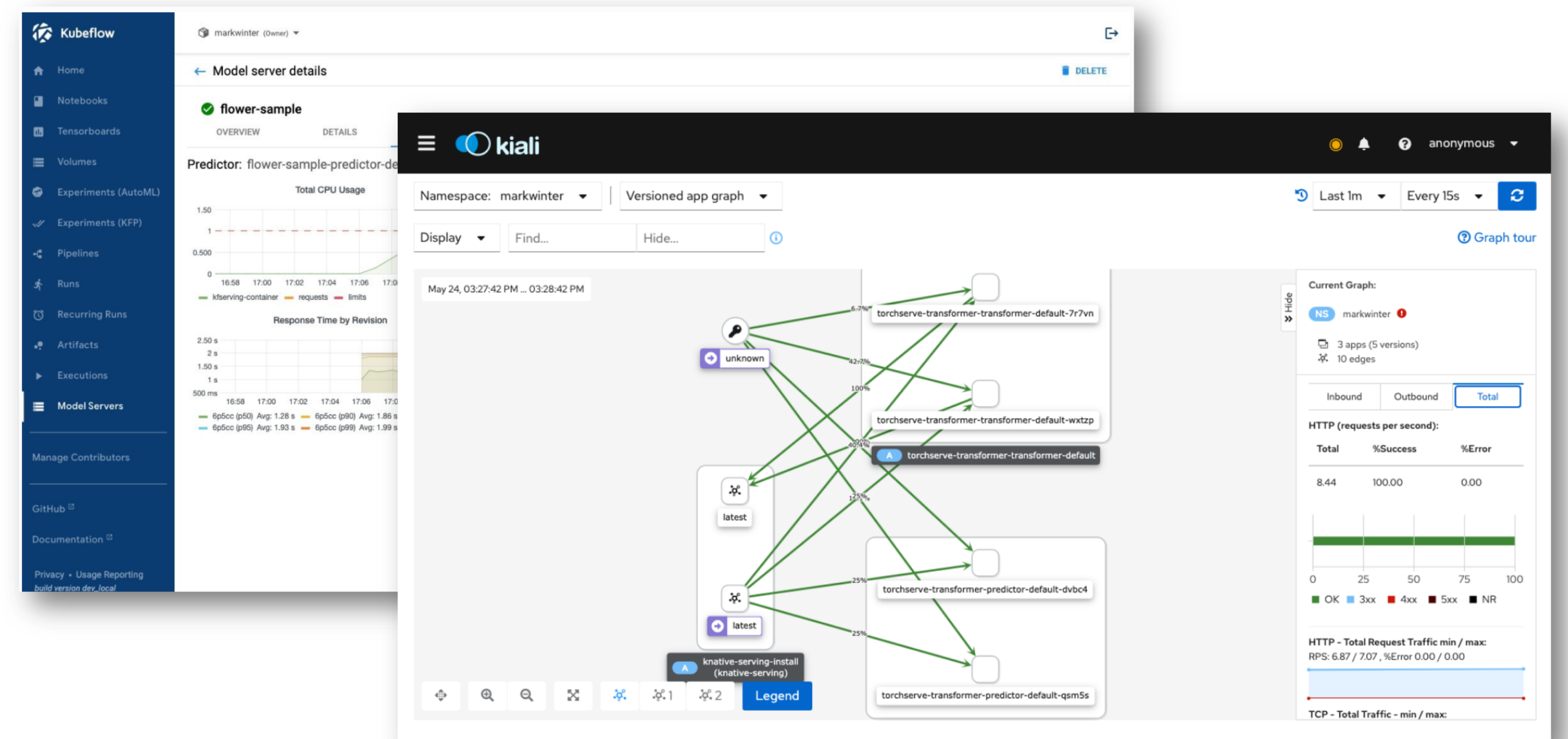
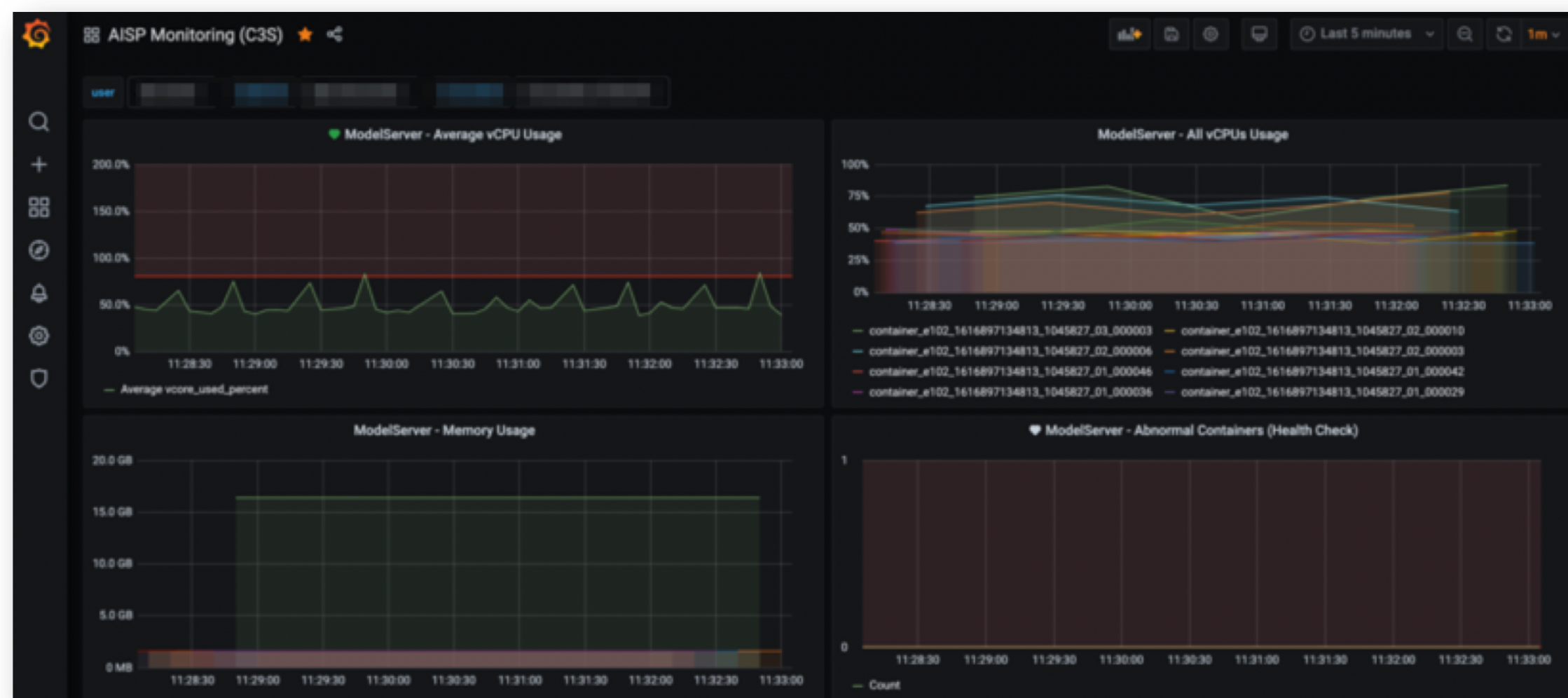
2.2 기존 서빙 플랫폼과의 비교

모니터링

사내 플랫폼 별 모니터링 시스템에 의존
AiSP 클라이언트 라이브러리에서 직접 구현

AiSP | KFServing

다양한 소스를 통한 모니터링 지표 제공
(Knative serving, Istio 서비스 메쉬 등)
Kiali : 서비스의 각종 지표 (트래픽 상황, 배포 상황 등) 들을 시각화하여 표현



2.2 기존 서빙 플랫폼과의 비교

서버 스케일링

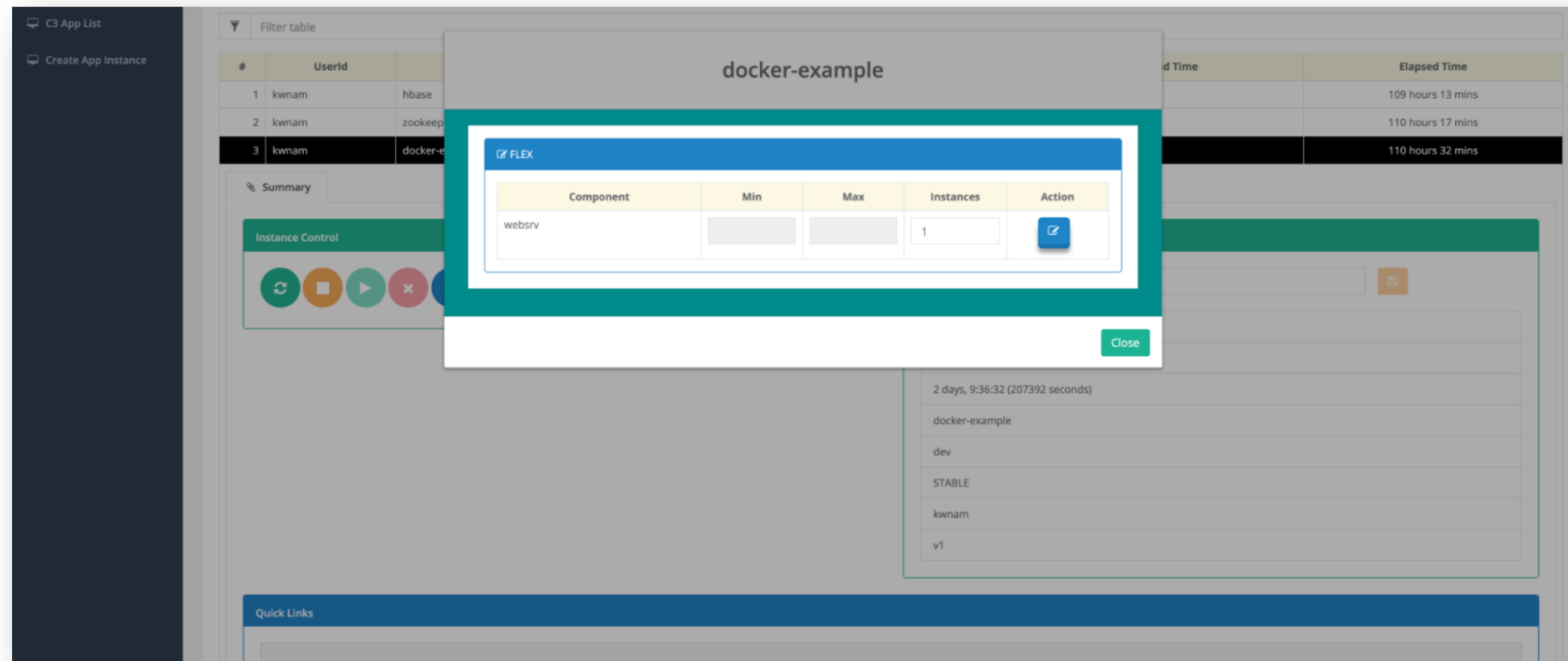
메뉴얼한 스케일링만 지원

AiSP

KFServing

CPU 사용량, 트래픽 기반 오토스케일링 지원

Knative 제로 스케일 지원



```
apiVersion: "serving.kubeflow.org/v1beta1"
kind: InferenceService
metadata:
  name: sentiment
  annotations:
    autoscaling.knative.dev/target: "150"
    autoscaling.knative.dev/metric: "rps"
spec:
  predictor:
    onnx:
      storageUri: ""
```


2.2 기존 서빙 플랫폼과의 비교

배포

블루/그린 배포

=> 2배 리소스 필요

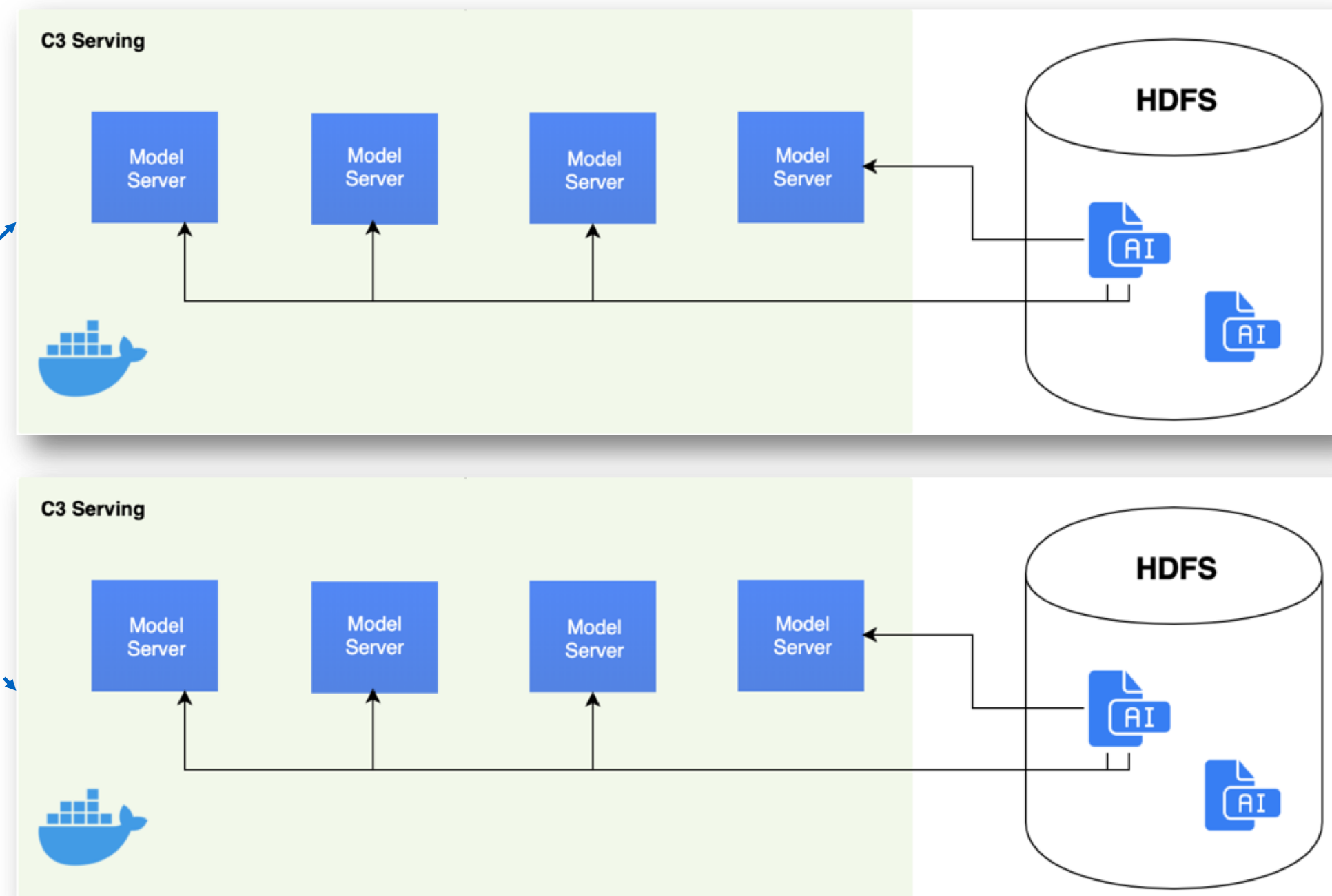
AiSP

KFServing

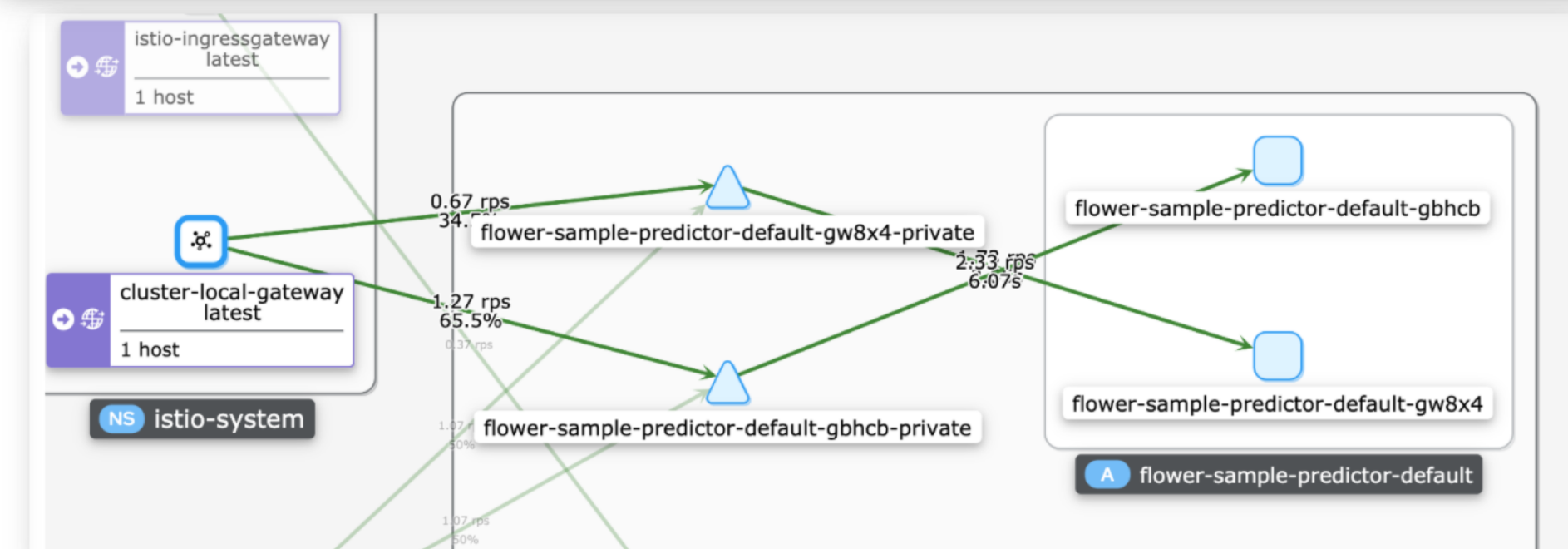
카나리 배포

=> 추가 리소스 필요 없음

=> 실트래픽 테스트를 통한 안정적인 배포 가능



```
# tensorflow.yaml
apiVersion: "serving.kubeflow.org/v1beta1"
kind: "InferenceService"
metadata:
  name: "flower-sample"
spec:
  predictor:
    canaryTrafficPercent: 5
    tensorflow:
      storageUri: "gs://kfserving-samples/models/tensorflow/flowers-v2"
```



2.2 기존 서빙 플랫폼과의 비교

정리하자면, KFServing을 사용하면

유저 입장에서는

- 단일 진입점으로 요청하면 되어 편리해진다. 프로그래밍 언어의 제약이 사라진다.
- 좀 더 다양한 지표를 더 다양한 방법으로 시각화하여 모니터링 할 수 있다.

운영자 입장에서는

- 프로그래밍 언어별로 라이브러리를 개발, 유지보수하는 비용이 줄어든다.
- 오토스케일링, 카나리 배포를 통해 자원을 아낄 수 있고, 운영 안정성이 증가한다.

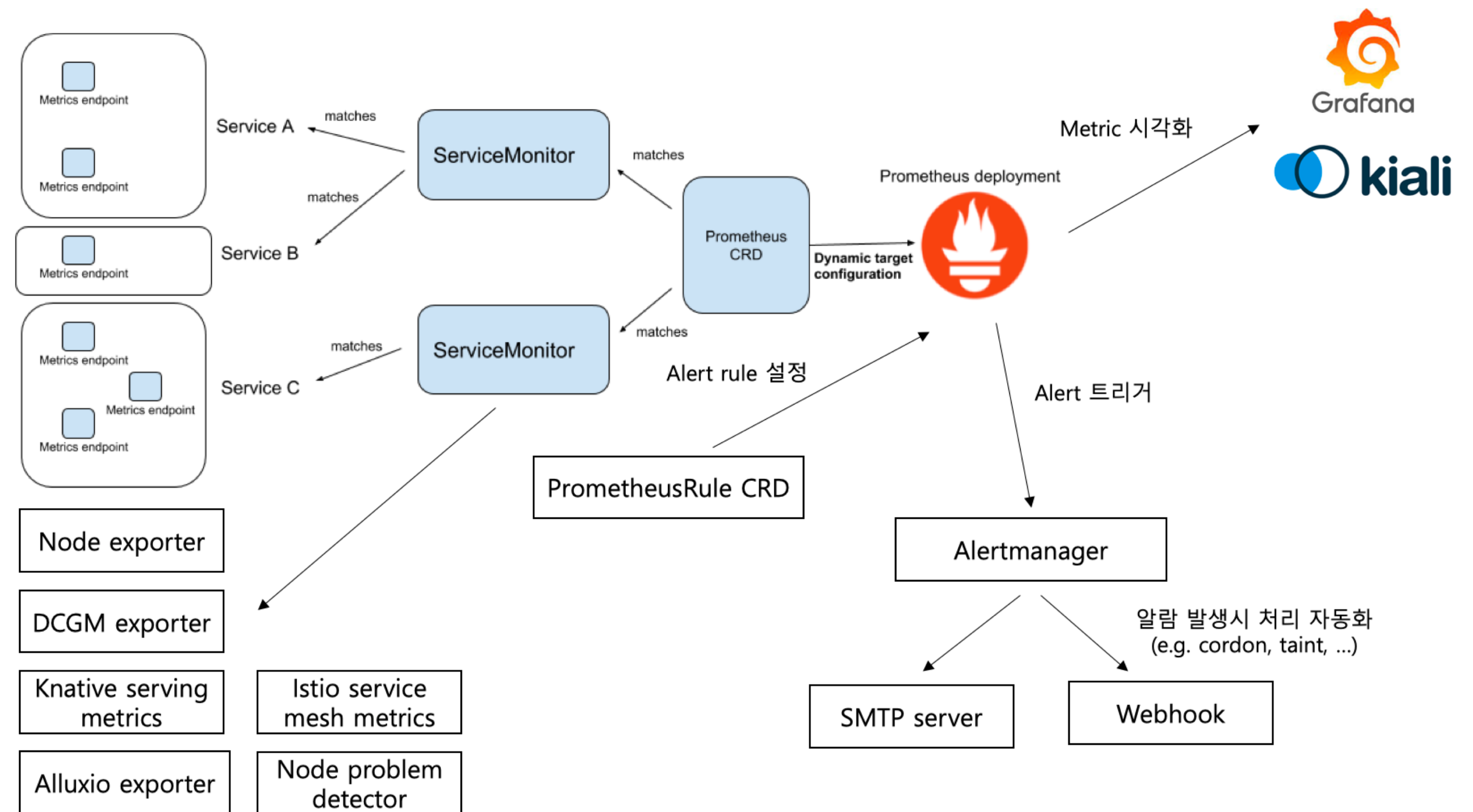
2.3 KFServing 환경 구축하기

Prometheus 를 통한 모니터링 지표 수집

- Prometheus Operator : ServiceMonitor 와 PodMonitor라는 CRDs 통해서 손쉽게 모니터링 타겟을 설정하도록 지원

모니터링 타겟

- kube-state-metrics : k8s API 서버에서 k8s 오브젝트 지표 수집
- metrics-servers : kubelet에서 노드, 컨테이너 지표 수집
- DCGM exporter : GPU 지표 수집
- Knative serving metrics : Knative 지표 수집
- Istio service mesh metrics : Istio 지표 수집

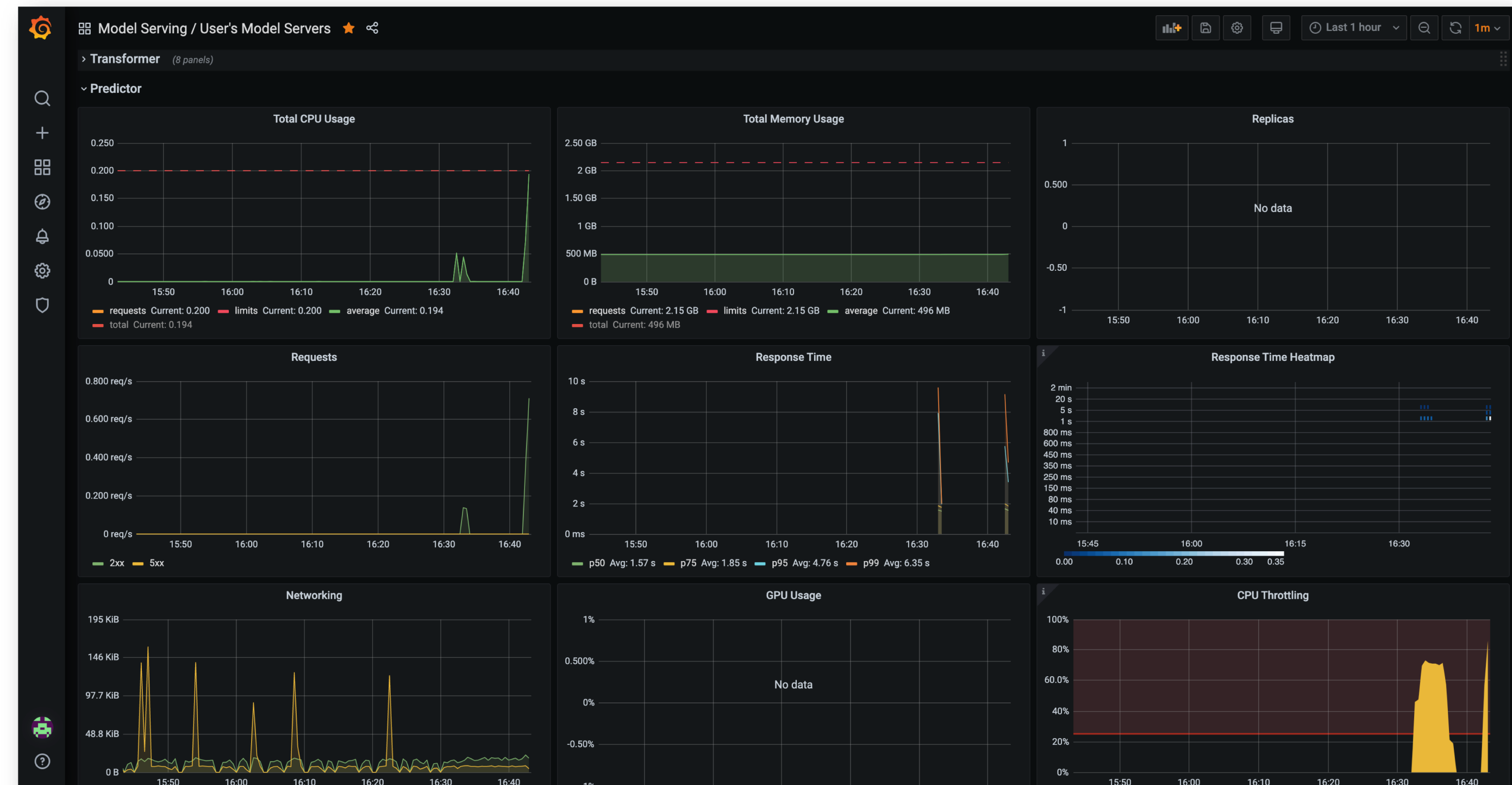


2.3 KFServing 환경 구축하기

Grafana 커스텀 대시보드 구성

- CPU, 메모리, GPU, 네트워크 사용량, 처리량, 응답시간, CPU throttling 와 같은
지표를 한눈에 볼 수 있게 제공

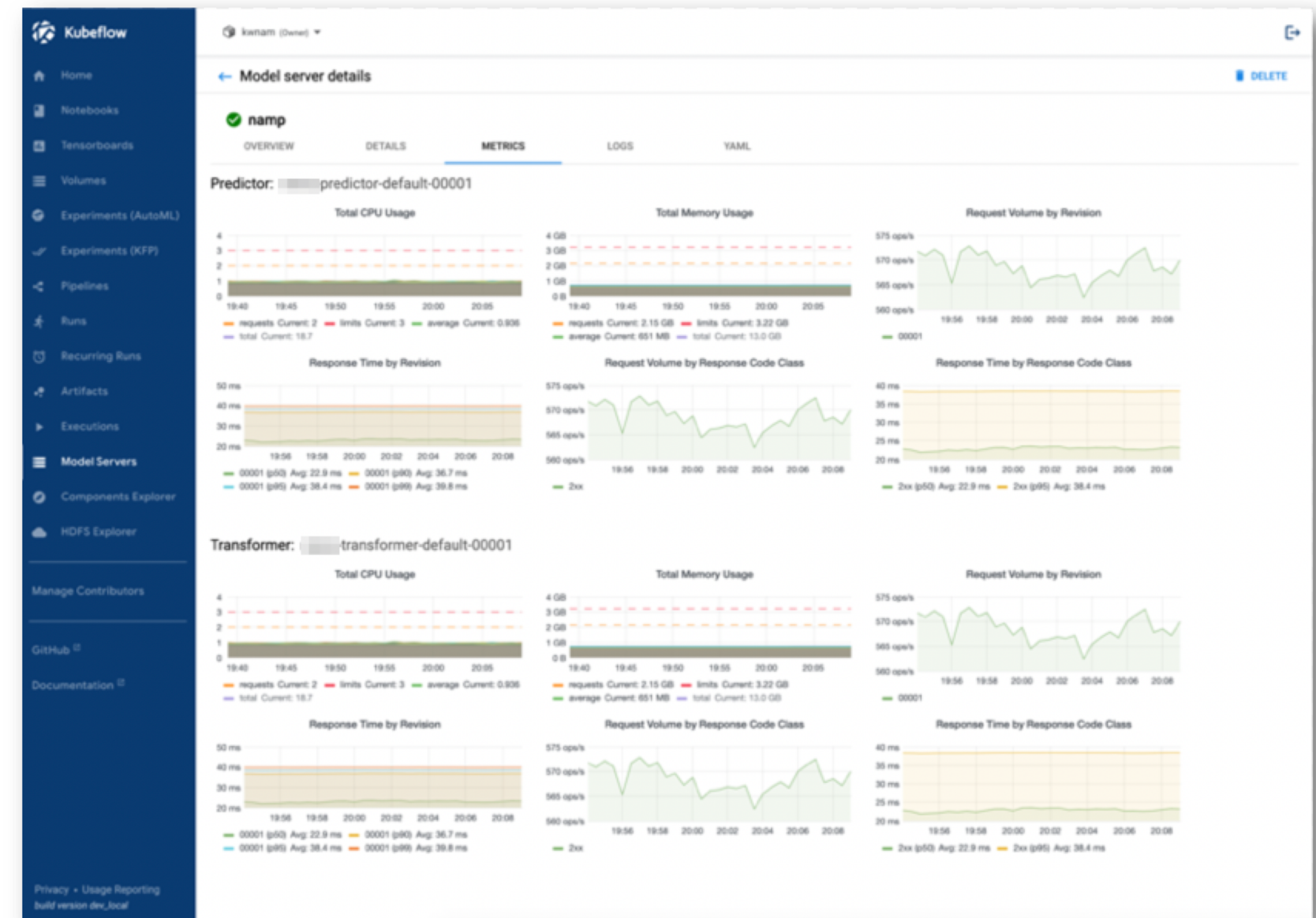
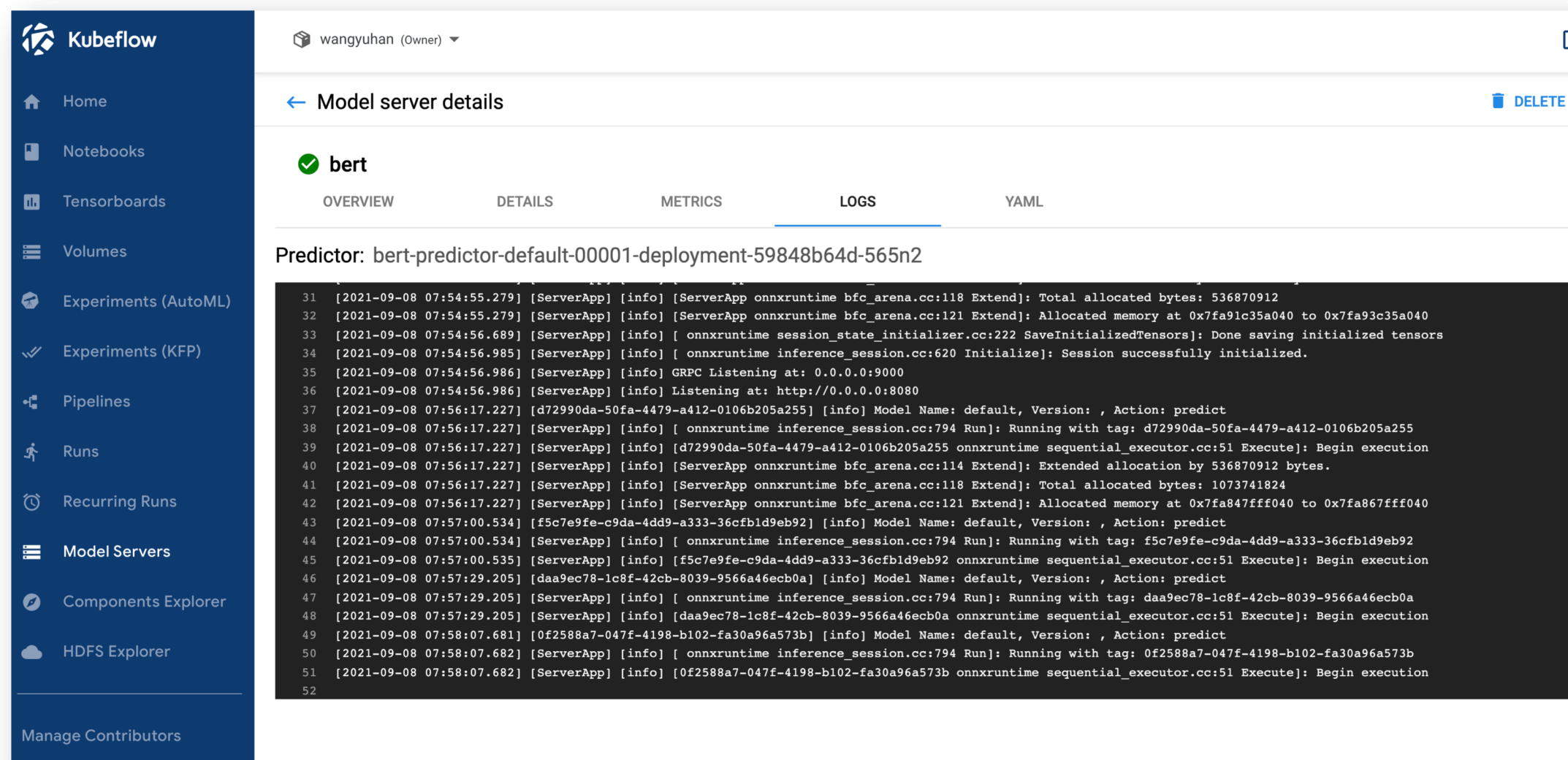
- metrics-server, kube-state-metrics,
Istio, Knative serving, DCGM exporter
와 같은 여러 모니터링 소스로부터의
지표를 조합하여 구성



2.3 KFServing 환경 구축하기

Kubeflow 대쉬보드 모니터링 구성

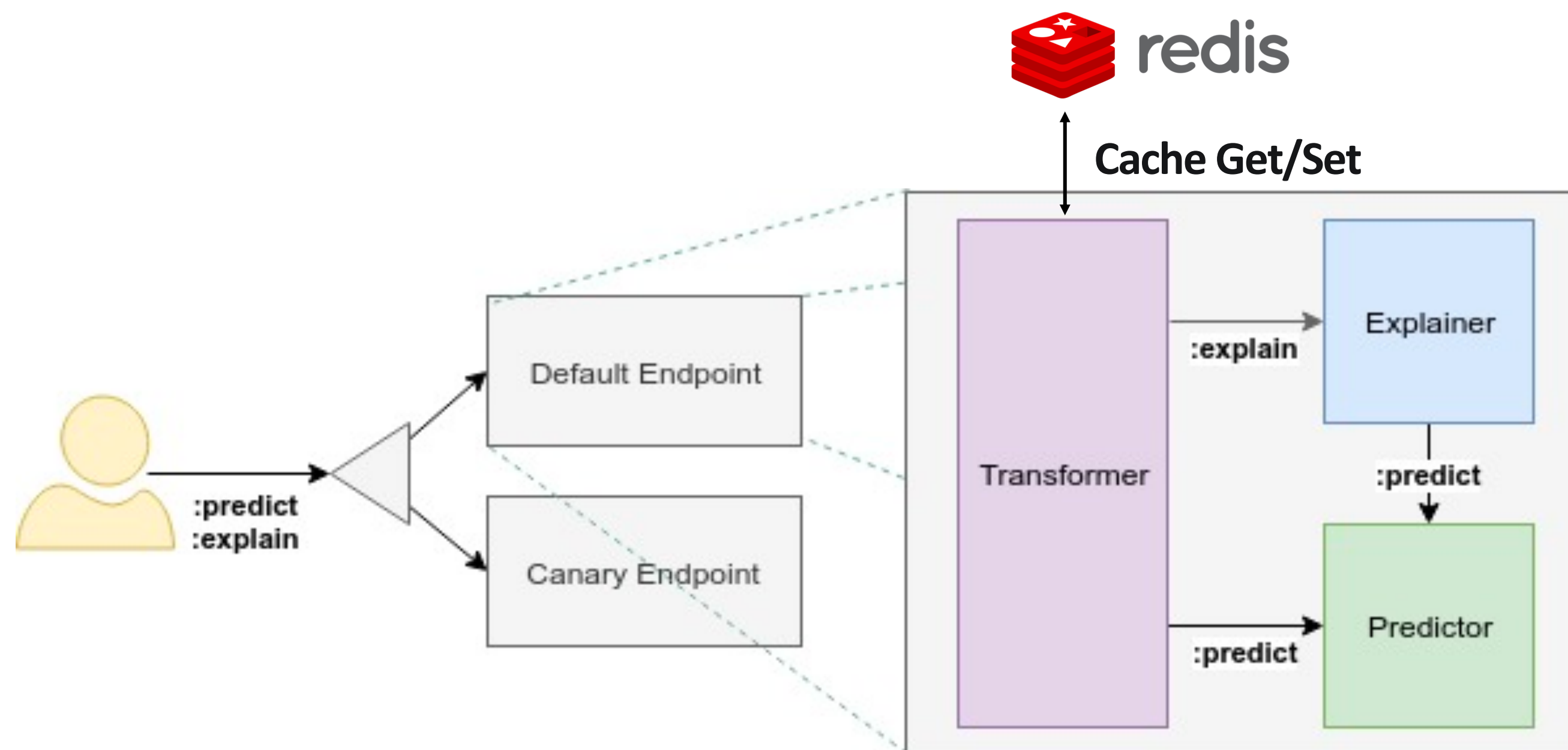
- Kubeflow 대쉬보드를 통해 서버의 리소스, 트래픽 상태 뿐만 아니라 오브젝트 상태, 로그, YAML 에 대한 정보를 제공
- KFServing 버전 0.6으로 업그레이드 필요



2.3 KFServing 환경 구축하기

캐싱

- 결과를 캐싱하여 응답시간을 줄임 (ex) 네이버 쇼핑 딥러닝 모델 서빙에 적용시 평균 응답시간이 50% 감소)
- 서비스 별로 독립적인 Redis 서버를 사용하도록 구성

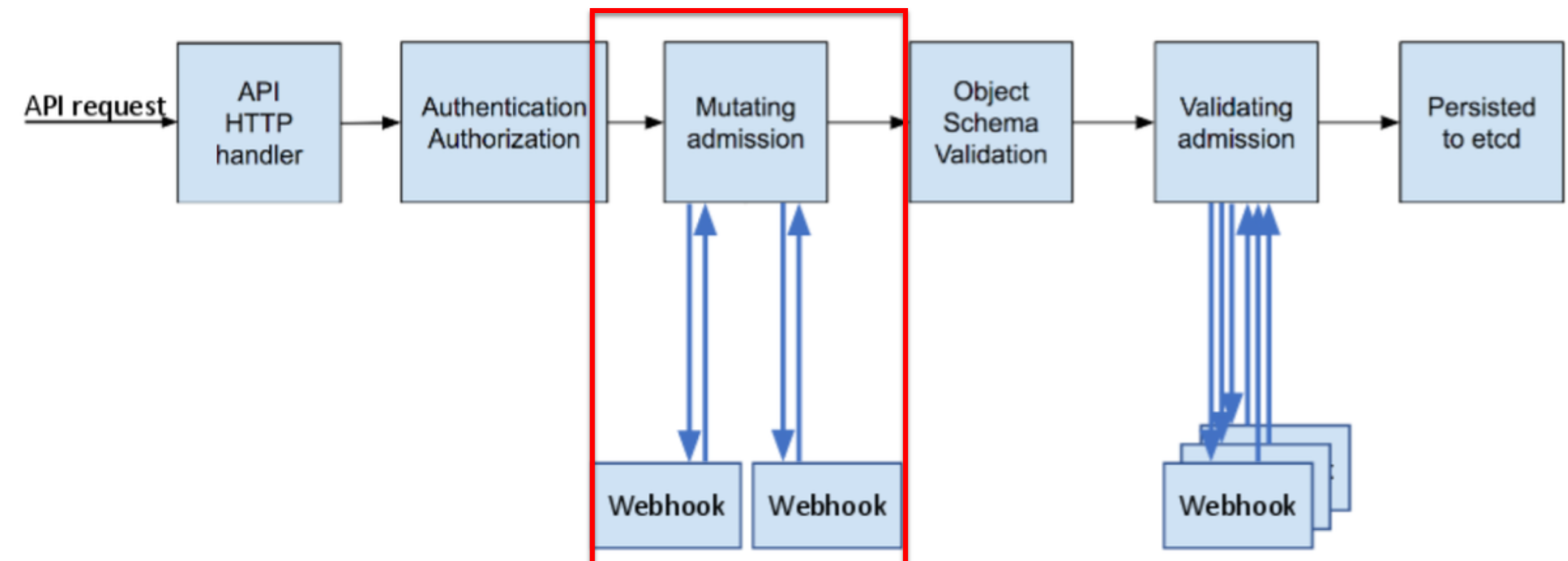


2.3 KFServing 환경 구축하기

캐싱 구성 과정

- InferenceService CRD에서 사이드카 컨테이너 설정을 할 수 없는 문제
- annotation을 붙이면, 어드미션 웹훅을 통해 컨테이너 생성 요청시 Redis를 사이드카 컨테이너로 동작시키도록 구현
- Mutating Admission Webhook : Admission controllers에서 Kubernetes API 서버로 가는 요청을 중간에 가로채는 웹훅을 만들어 해당 요청을 변환

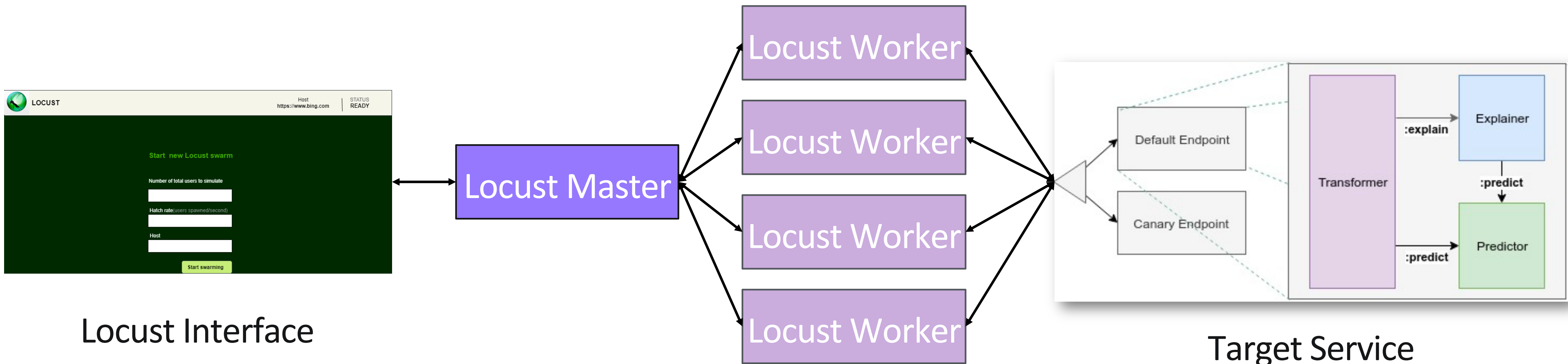
```
apiVersion: serving.kubeflow.org/v1beta1
kind: InferenceService
metadata:
  name: torchserve
  namespace: markwinter
  annotations:
    /transformer-cache: "true"
    /transformer-cache-config: redis-config
```



2.4 실서비스 성능 테스트

네이버 쇼핑에서 사용되는 딥러닝 모델로 성능 테스트 진행

- AiSP와 동일한 전/후 처리 로직을 동일한 언어 (C++) 로 구현
- 분산 부하 테스트 툴 Locust 로 실서비스와 동일한 트래픽을 가해 응답시간 측정



2.4 실서비스 성능 테스트

기존 플랫폼과 비교해 성능 저하를 우려했으나...

부하 테스트 결과, 동일한 서버 자원으로 평균 응답시간이 동일함!



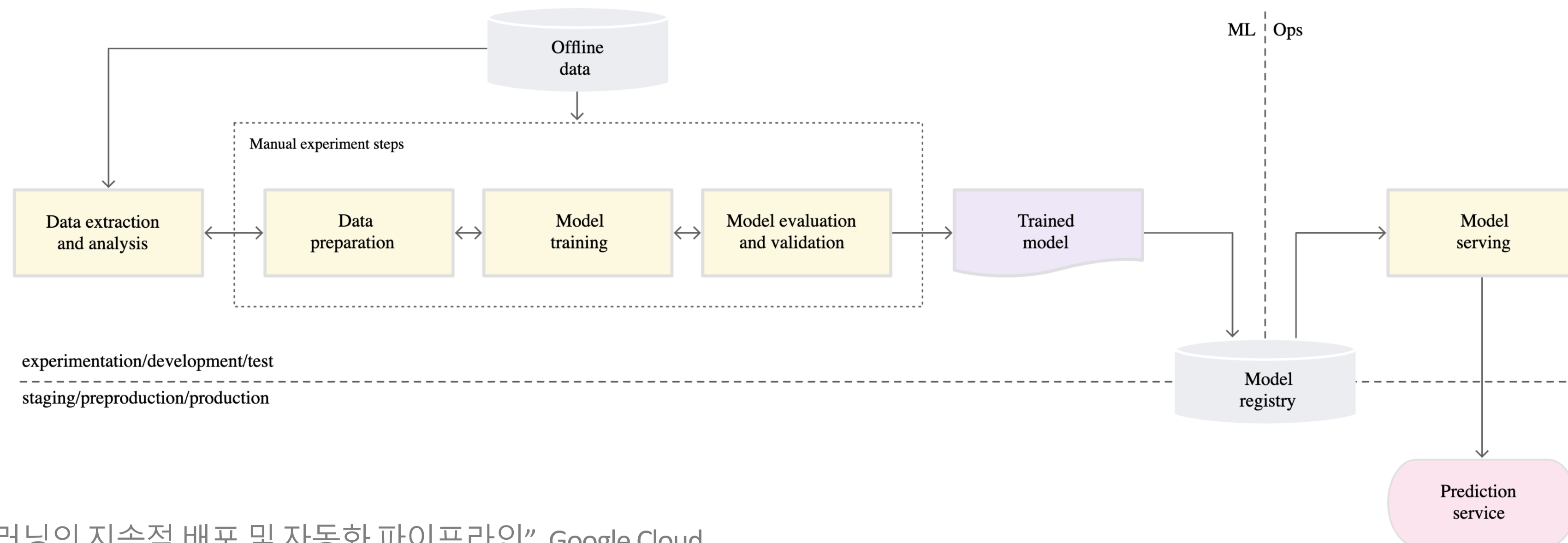
3. Kubeflow로 MLOps 실현하기

3.1 MLOps 란?

MLOps 수준 0: 수동 프로세스

- 데이터 준비, 모델 학습, 모델 평가가 수동으로 이루어짐
- 모델을 플랫폼 운영팀에 전달해 배포 진행

=> 기존 플랫폼에서 지원하던 수준



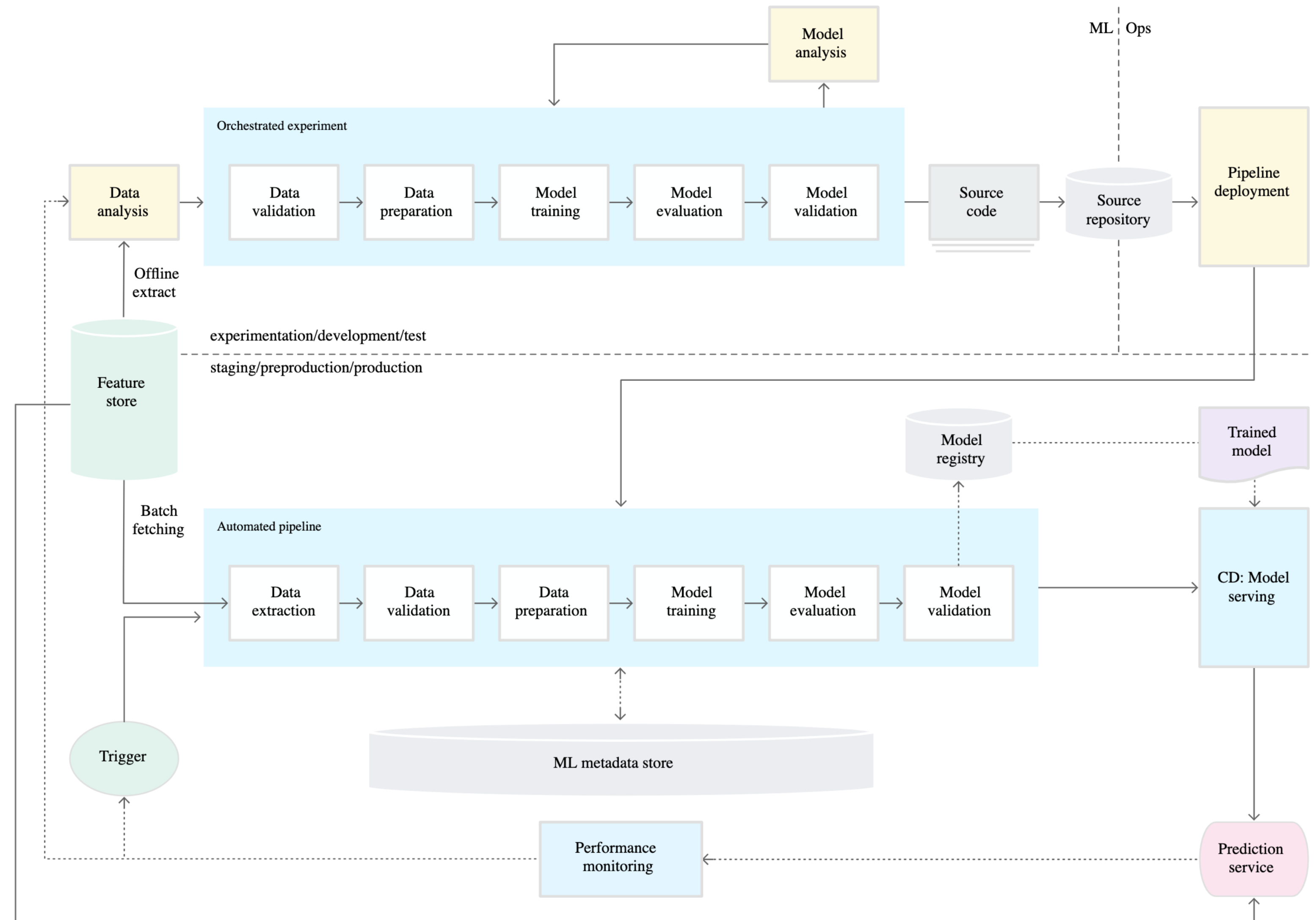
3.1 MLOps 란?

MLOps 수준 1: ML

파이프라인 자동화

- 데이터 준비, 모델 학습, 모델 평가 파이프라인 자동화
- 모델 배포 자동화
- 파이프라인 소스 수동 관리

=> Kubeflow 도입으로 많은 부분이 지원됨

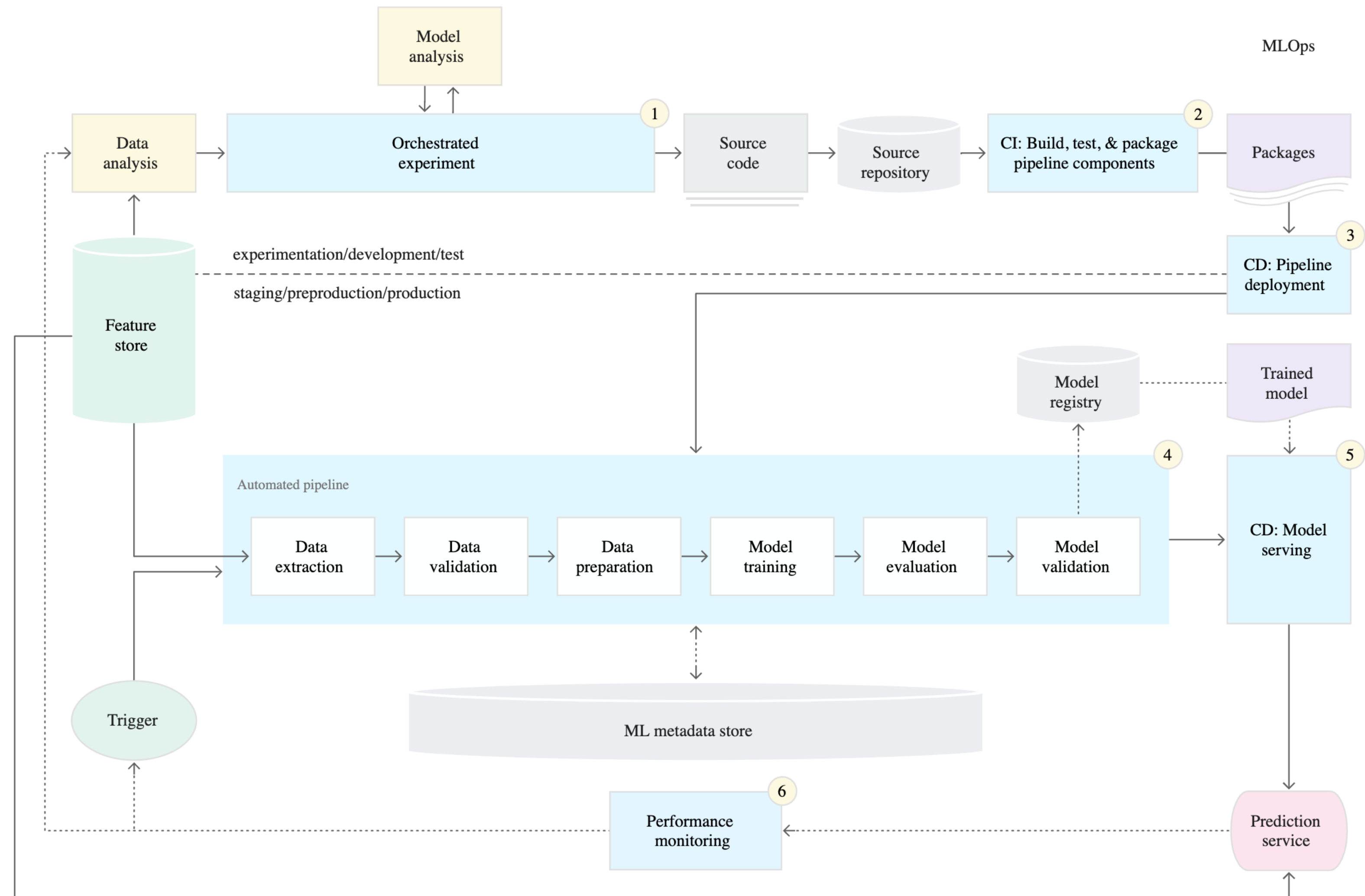


3.1 MLOps 란?

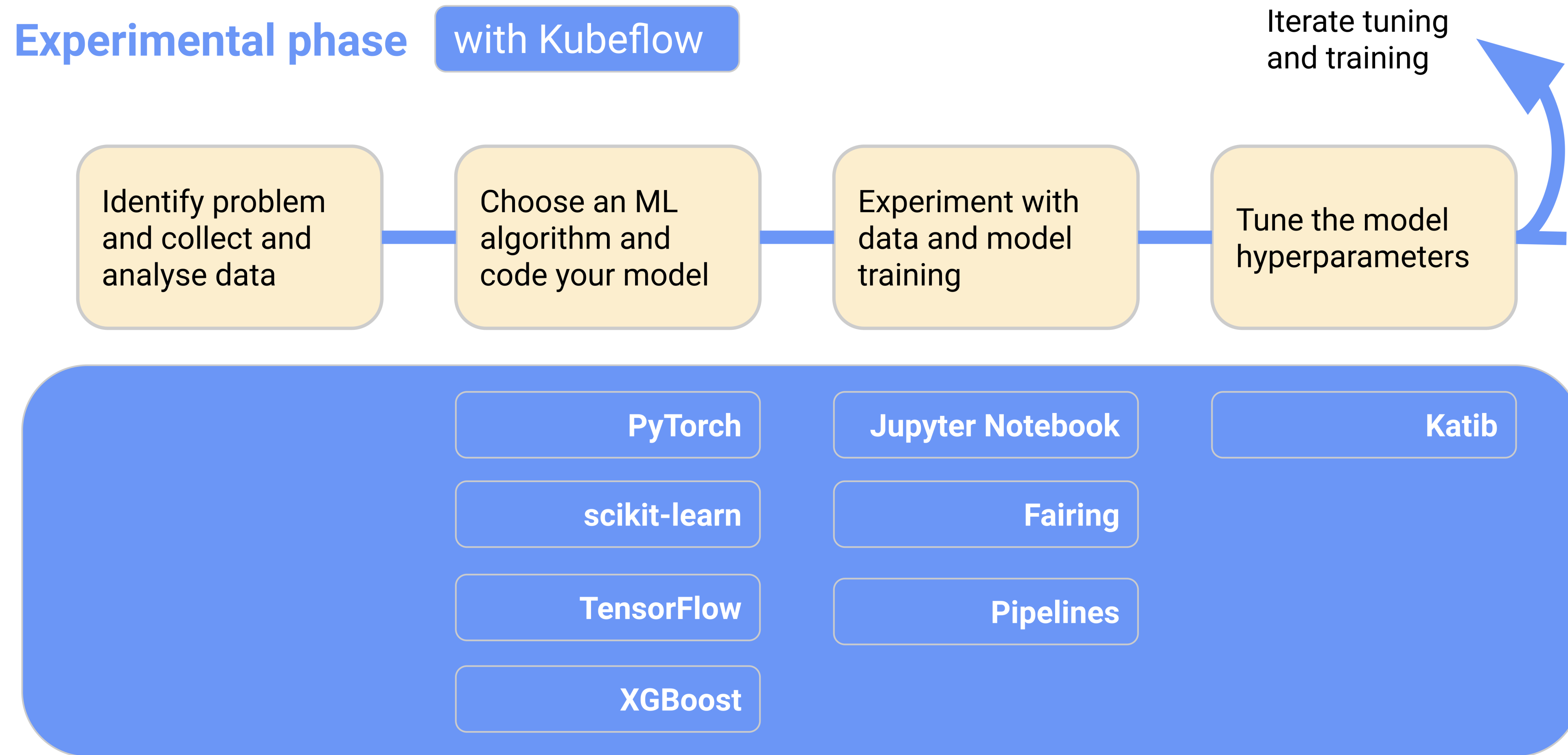
MLOps 수준 2: CI/CD

파이프라인 자동화

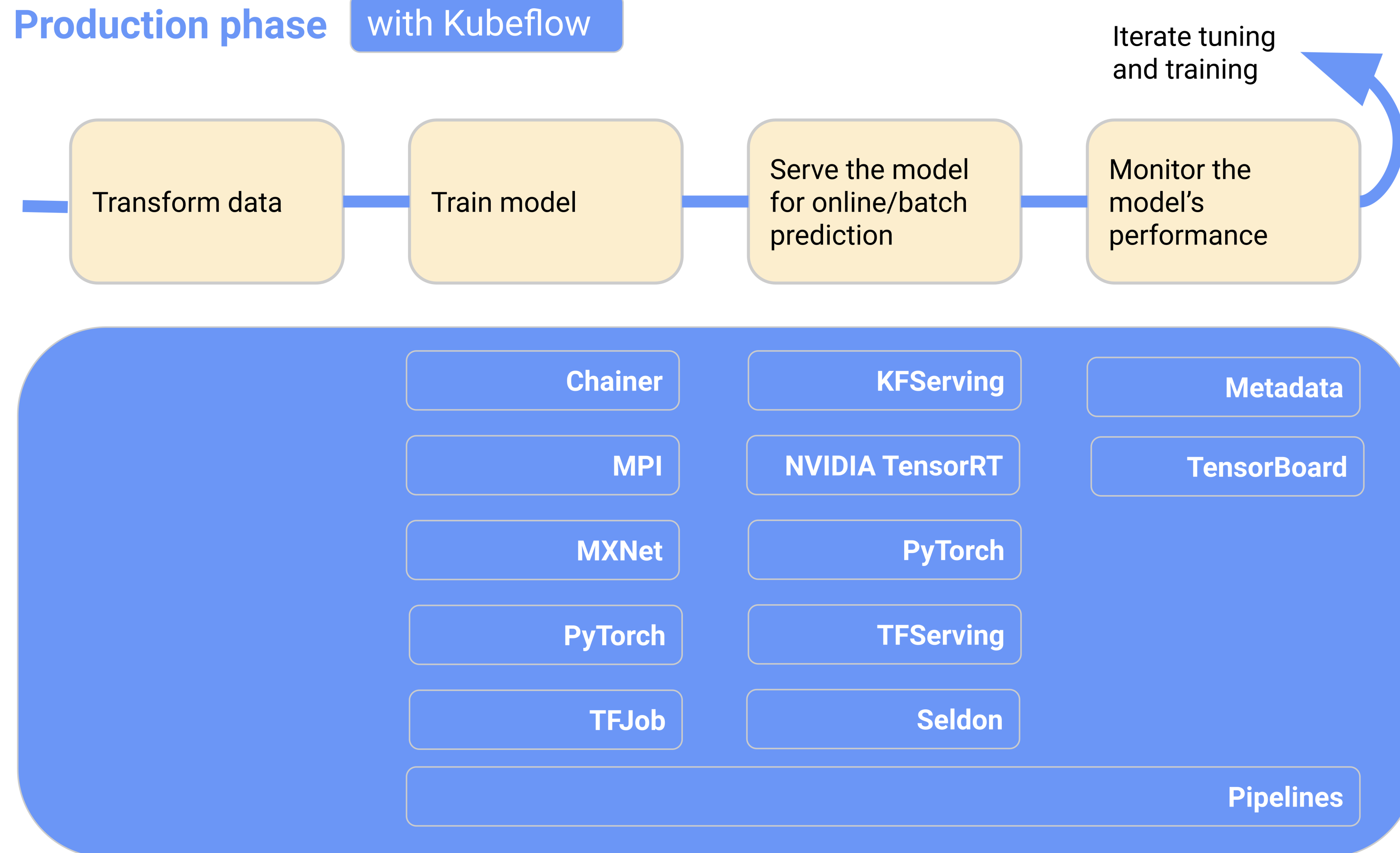
- 파이프라인 소스 관리 자동화
- 모델 단위 테스트 자동화
- 배포 안정성 테스트 자동화
(인프라 호환성, 실트래픽 검증)



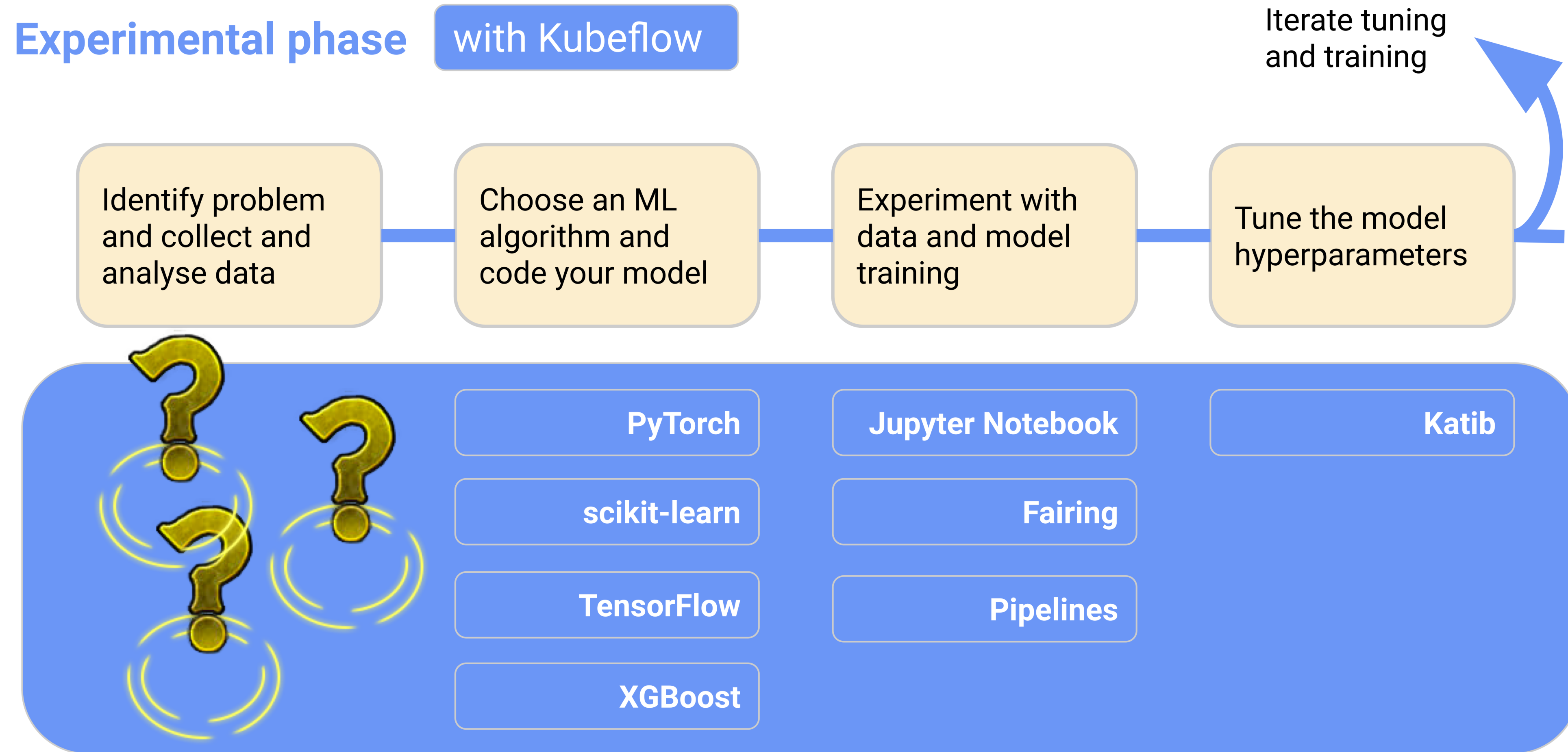
3.2 Kubeflow 컴포넌트



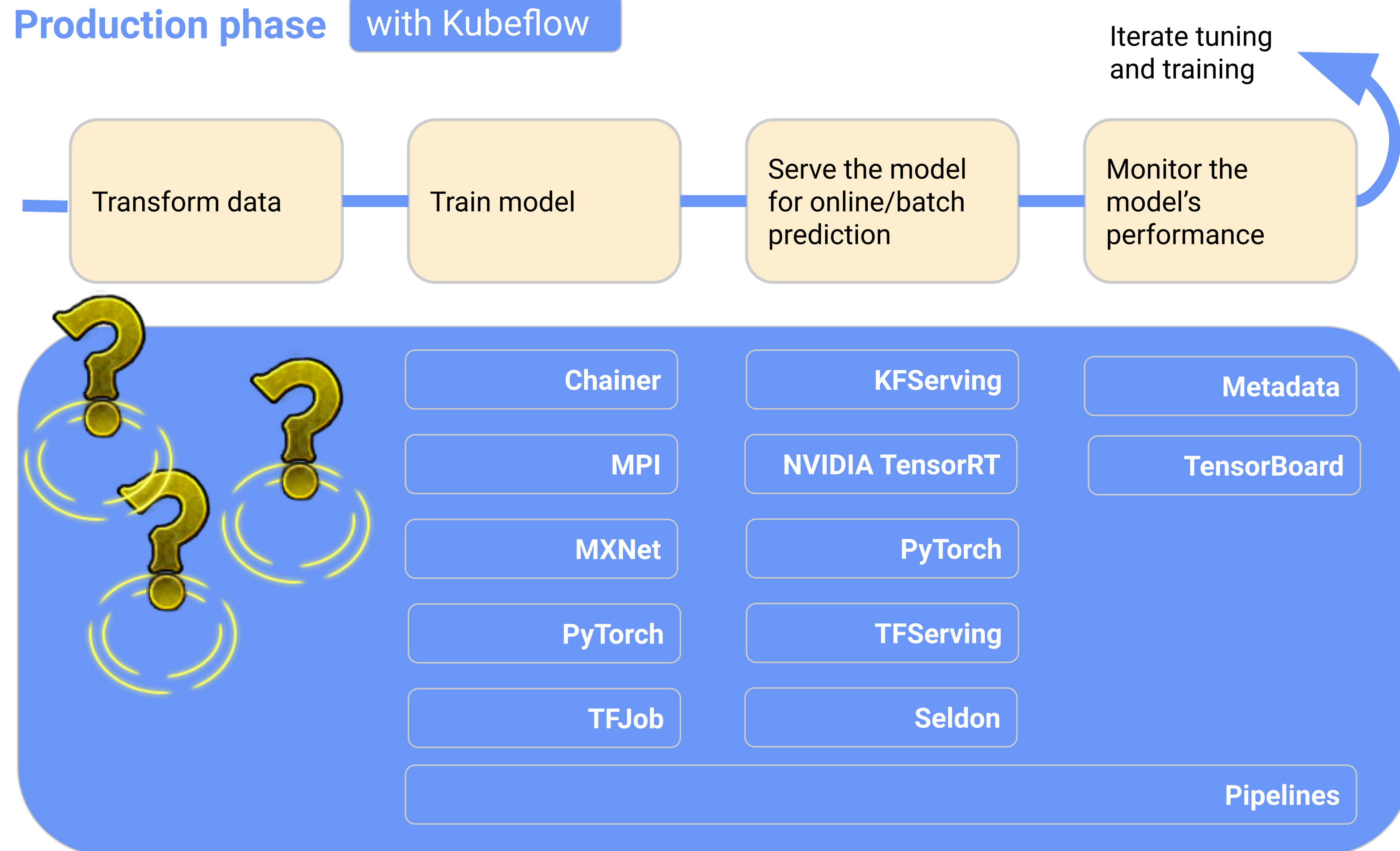
3.2 Kubeflow 컴포넌트



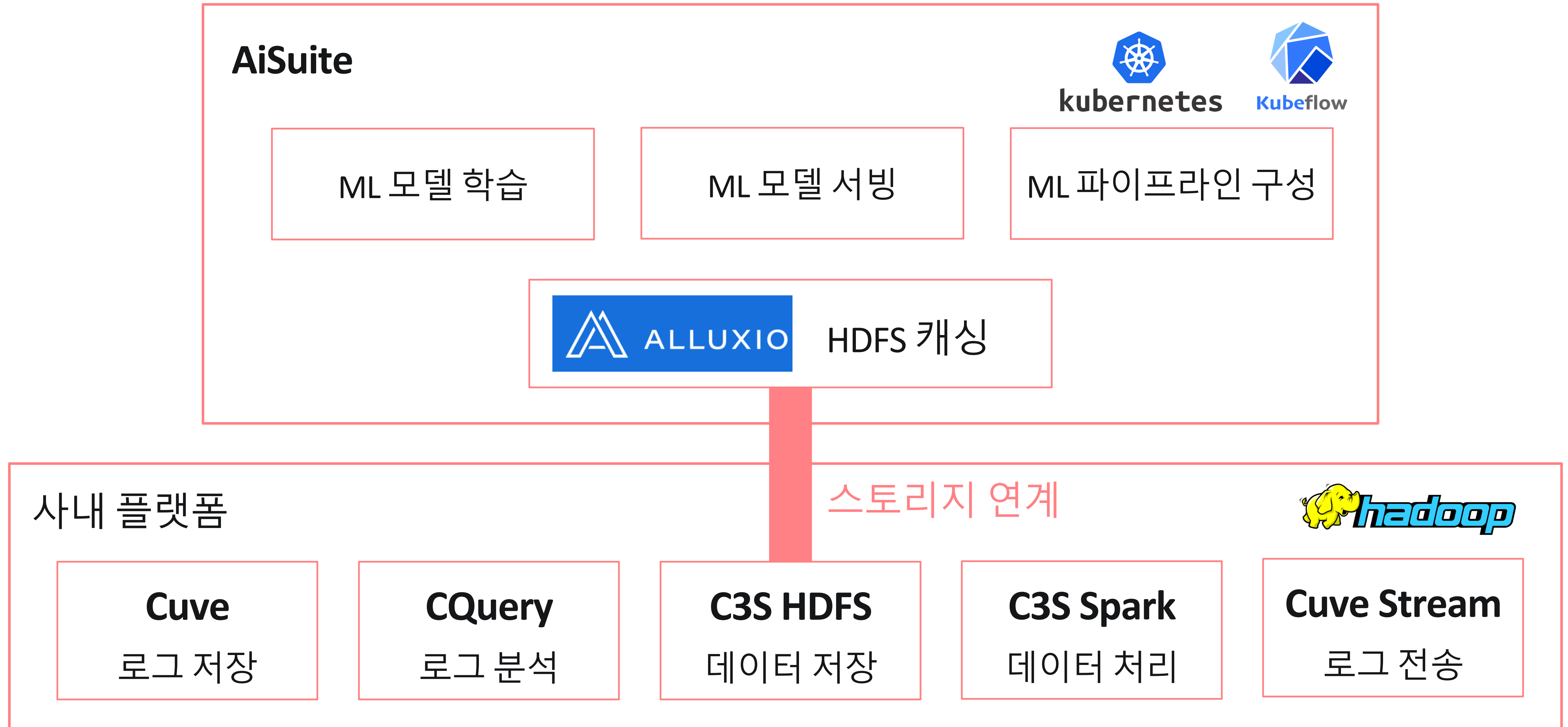
3.2 Kubeflow 컴포넌트



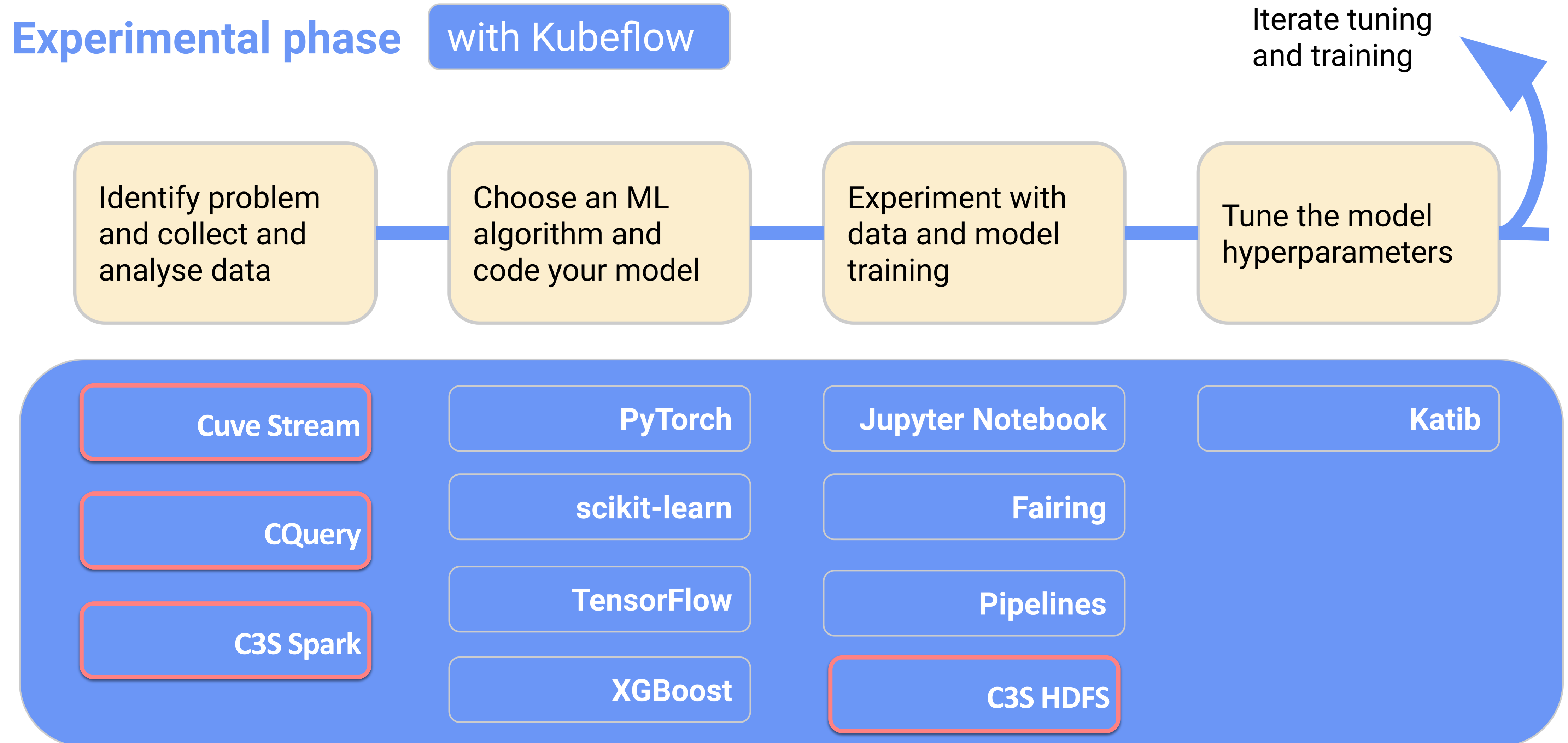
3.2 Kubeflow 컴포넌트



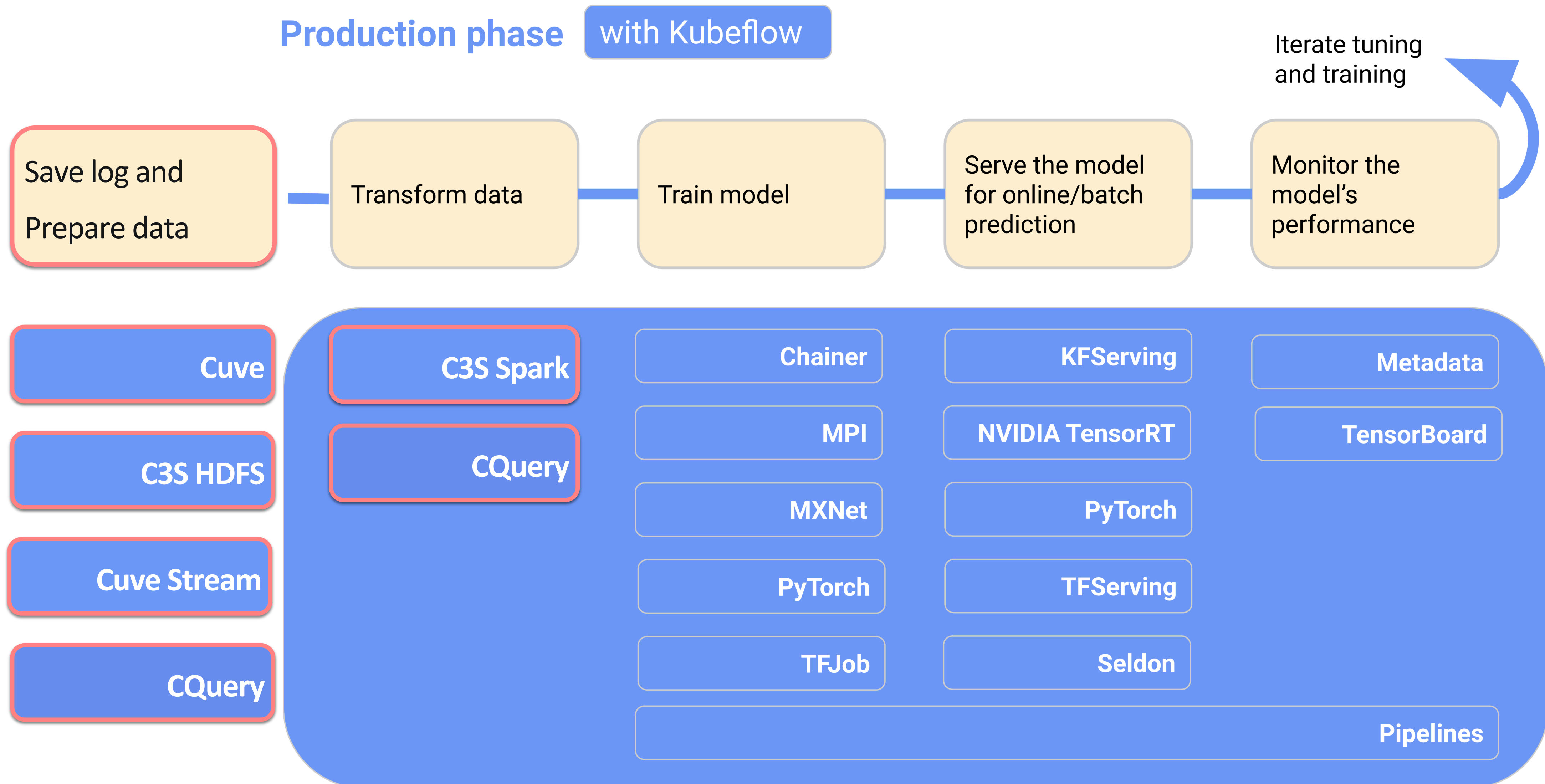
3.3 사내 플랫폼과의 연계



3.3 사내 플랫폼과의 연계



3.3 사내 플랫폼과의 연계



3.4 ML 파이프라인 구성

파이프라인 구성하기가 쉬워졌다!

1. 모듈 구현

- 파이프라인 모듈을 도커 이미지로 컨테이너화하여 구현

- 모듈의 입력, 출력을 명시하는 yml 파일 작성

2. 파이프라인 연결

- Pipelines SDK로 모듈들을 연결

master
kubeflow-pipeline-components / components / cuve-transfer /
Go to file
Add file

mark-winter Fixes

f170940 on Aug 30
History

..		
src	Fixes	last month
Dockerfile	Pull specific tag	2 months ago
README.md	Remove references to docker secret	3 months ago
build_image.sh	Add substitutions for e2e component	last month
component.e2e.yaml	Bump version	last month
component.inline.yaml	bump version	2 months ago

README.md

cuve-transfer

Transfer data from Cuve to HDFS using PyLagos library

```

11
12
13
14
15
16
17
18 hdfs_save_path: str,
19 transfer_queue: str,
20 ):
21     """Transfer data selected with SQL to specified HDFS save path."""
22     conn = hive.connect(
23         principal=[REDACTED],
24         keytab_file=keytab_path,
25         location=cuve_location,
26         ticket=cuve_ticket,
27         config={REDACTED},
28     )
29
30     cursor = conn.cursor()
31     with open(cuve_sql, "r") as sql_file:
32         cursor.save_hdfs(sql_file.read(), cluster_name=hdfs_namespace, path=hdfs_save_path)
33     cursor.close()
34     conn.close()

```


3.4 ML 파이프라인 구성

파이프라인 구성하기가 쉬워졌다!

1. 모듈 구현

- 파이프라인 모듈을 도커 이미지로
컨테이너화하여 구현

- 모듈의 입력, 출력을 명시하는

yaml 파일 작성

2. 파이프라인 연결

- Pipelines SDK로 모듈들을 연결

```
1  name: Cuve Transfer
2  description: Transfer data from Cuve to HDFS using SQL
3  inputs:
4    - {name: Cuve Location}
5    - {name: Cuve SQL}
6    - {name: C3S Username}
7    - {name: HDFS Namespace}
8    - {name: HDFS Save Path}
9    - {name: Transfer Queue}
10 outputs:
11   - {name: HDFS Path}
12 implementation:
13   container:
14     image: 
15     command: [
16       python3, /kfp/component/src/transfer-inline.py,
17       --cuve-location, {inputValue: Cuve Location},
18       --cuve-sql, {inputPath: Cuve SQL},
19       --c3s-username, {inputValue: C3S Username},
20       --hdfs-namespace, {inputValue: HDFS Namespace},
21       --hdfs-save-path, {inputValue: HDFS Save Path},
22       --transfer-queue, {inputValue: Transfer Queue},
23       --output-path, {outputPath: HDFS Path},
24     ]
```

3.4 ML 파이프라인 구성

파이프라인 구성하기가 쉬워졌다!

1. 모듈 구현

- 파이프라인 모듈을 도커 이미지로 컨테이너화하여 구현
- 모듈의 입력, 출력을 명시하는 yaml 파일 작성

2. 파이프라인 구성

- Pipelines SDK로 모듈들을 연결

```
114 @kfp.dsl.pipeline(  
115     name='TFJob, HDFS PVC, KFServing E2E Pipeline',  
116 )  
117 def e2e_pipeline():  
118     # 데이터 준비 : Pylago를 이용해 Cuve에서 SQL로 데이터를 가져와 HDFS에 저장  
119     transfer_task = transfer_op(  
120         cuve_location='...',  
121         cuve_sql=sql,  
122         c3s_username='...',  
123         hdfs_namespace='...',  
124         hdfs_save_path='...',  
125         transfer_queue='...',  
126     ).apply(use_k8s_secret(secret_name='kerberos', k8s_secret_key_to_env={'keytab': 'KEYTAB'})) \  
127     .apply(use_k8s_secret(secret_name='cuve', k8s_secret_key_to_env={'ticket': 'TICKET'}))  
128     # Component caching 기능을 off  
129     transfer_task.execution_options.caching_strategy.max_cache_staleness = "P0D"  
130  
131     # 데이터 전처리 : C3S 클러스터에서 Spark를 이용해 데이터를 전처리  
132     sparkjob_task = sparkjob_op(  
133         spark_code=spark_code,  
134         requirements=requirements,  
135         c3s_username='...',  
136         hdfs_input=transfer_task.outputs["hdfs_path"],  
137         hdfs_output='...',  
138         queue='...',  
139     ).apply(use_k8s_secret(secret_name='kerberos', k8s_secret_key_to_env={'keytab': 'KEYTAB'}))  
140     sparkjob_task.execution_options.caching_strategy.max_cache_staleness = "P0D"  
141  
142     # HDFS PVC 생성 : C3S HDFS에 저장된 데이터 사용을 위해 Alluxio PVC 생성  
143     data_task = kfp.dsl.VolumeOp(  
144         name="Create Alluxio PVC",
```


3.4 ML 파이프라인 구성

파이프라인 구성하기가 쉬워졌다!

1. 모듈 구현

- 파이프라인 모듈을 도커 이미지로 컨테이너화하여 구현
- 모듈의 입력, 출력을 명시하는 yaml 파일 작성

2. 파이프라인 구성

- Pipelines SDK로 모듈들을 연결

=> 인프라 관리자가 아닌
사내 개발자 누구나 손쉽게
모듈을 만들고,
파이프라인으로 구성 가능!

3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

사내 플랫폼 연계 모듈 구현

데이터 준비 (Cuve, CQuery),

데이터 전처리 (C3S Spark)

README.md

cuve-transfer

Transfer data from Cuve to HDFS using PyLagos library

```
usage: transfer.py [-h] --cuve-location CUVE_LOCATION --cuve-sql CUVE_SQL
                  --c3s-username C3S_USERNAME --hdfs-namespace HDFS_NAMESPACE
                  --hdfs-save-path HDFS_SAVE_PATH --transfer-queue
                  TRANSFER_QUEUE
```

Cuve-HDFS data transfer utility

optional arguments:

README.md

SparkJob

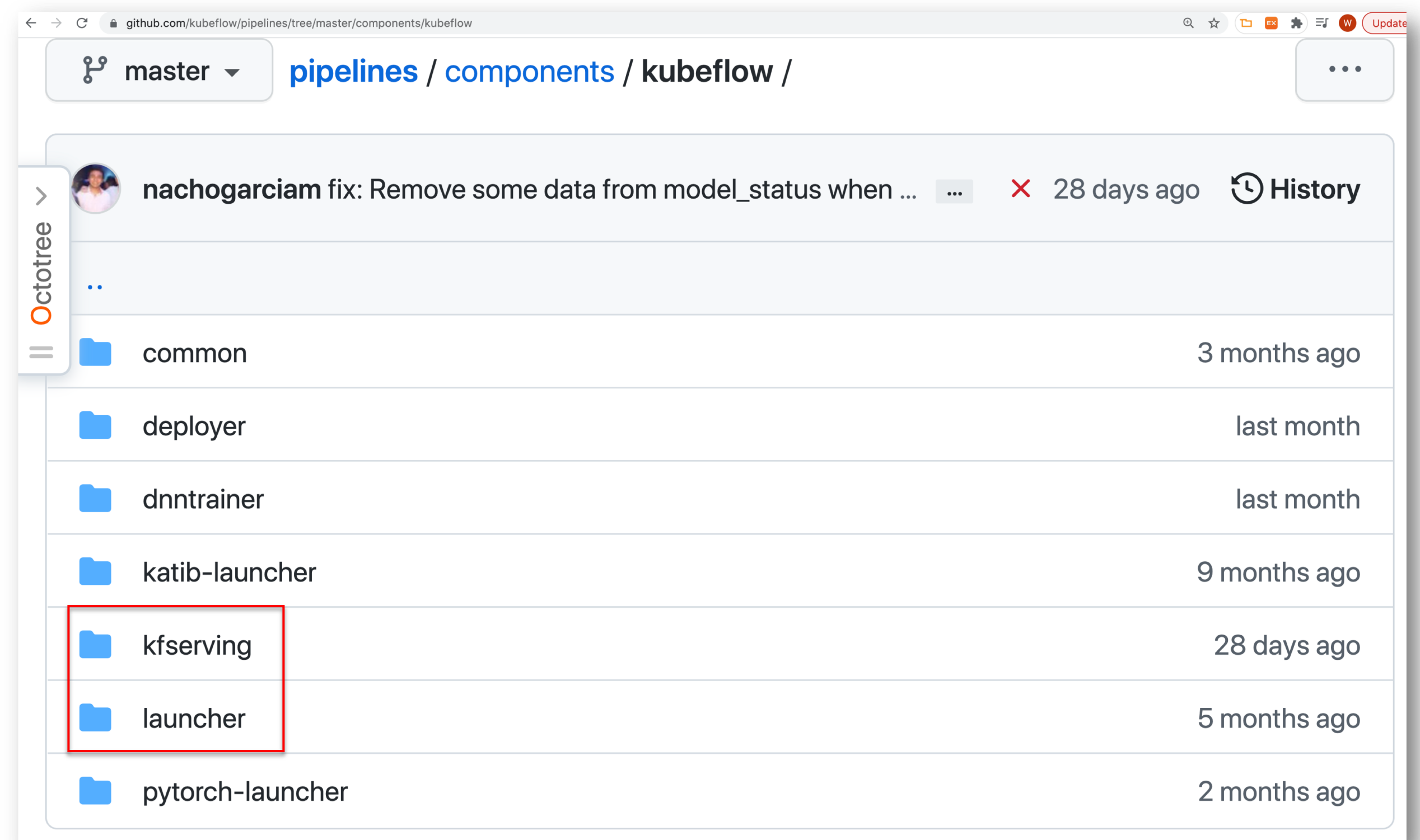
Run a Spark job on C3S using code from a Git repo

```
→ sparkjob git:(master) x src/spark_submit-inline.sh
Usage: ./spark_submit-inline.sh
      -f <SPARK CODE FILE>
      -r <PYTHON REQUIREMENTS FILE>
      -i <HDFS INPUT PATH>
      -o <HDFS OUTPUT PATH>
      -u <C3S USERNAME>
```

3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

Kubeflow 컴포넌트들에 대한 모듈들은 제공되는 것을 사용

HDFS PVC 생성 (C3S HDFS),
모델 학습 (TFJob),
모델 서빙 (KFServing)



3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

파이프라인 구성

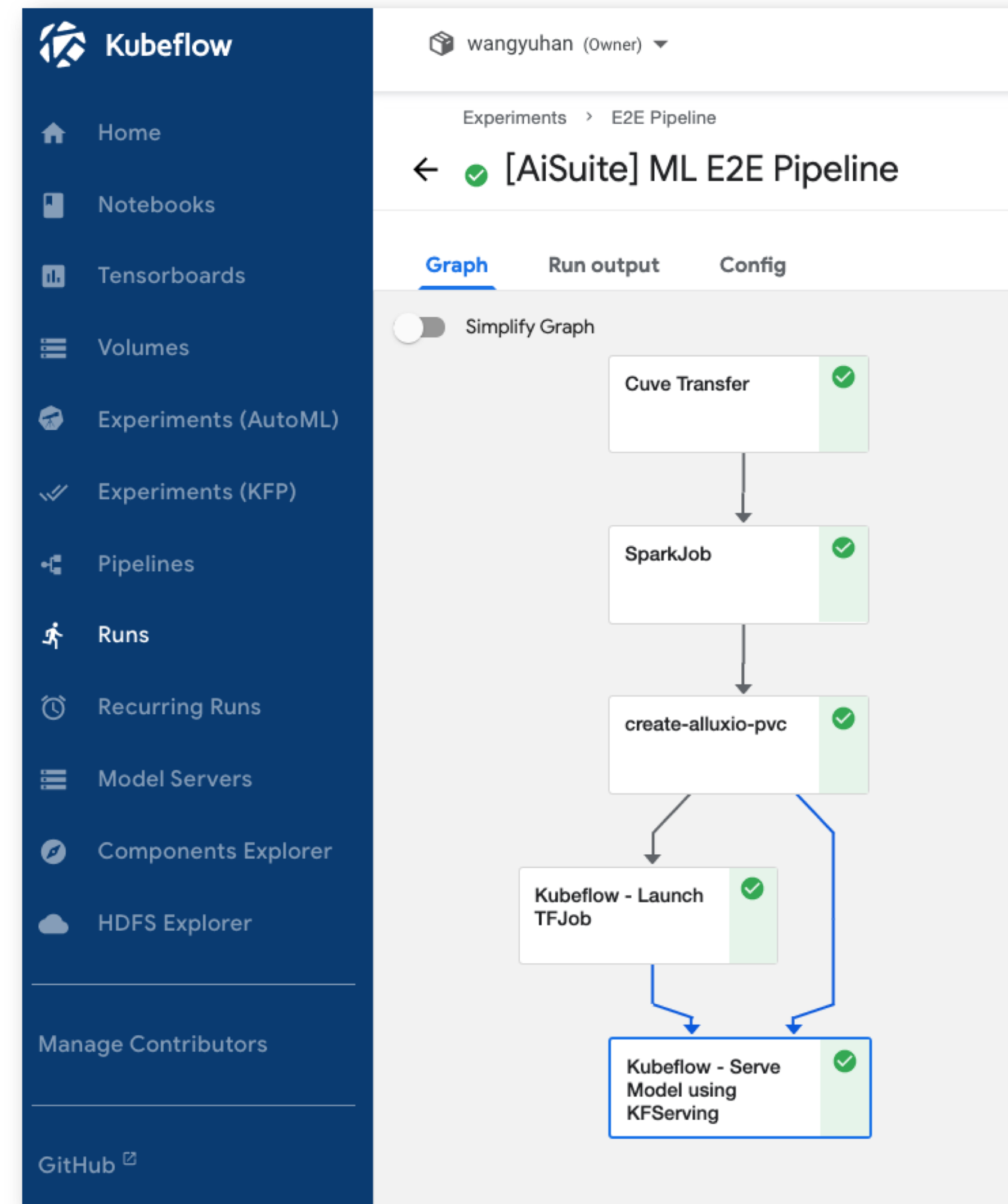
데이터 준비 (Cuve, CQuery) ->

데이터 전처리 (C3S Spark) ->

HDFS PVC 생성 (C3S HDFS) ->

모델 학습 (TFJob) ->

모델 서빙 (KFServing)



3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

데이터 준비 (Cuve, CQuery) ->

데이터 전처리 (C3S Spark) ->

HDFS PVC 생성 (C3S HDFS) ->

모델 학습 (TFJob) ->

모델 서빙 (KFServing)

```
14 ##### Cuve Transfer #####
15 sql = """
16 SELECT raw, label, type
17 FROM korea_ai_public_dataset.mnist
18 WHERE cuve_start_time="201211291900" and cuve_end_time="201211292000"
19 limit 30
20 """
```

```
114 @kfp.dsl.pipeline(
115     name='TFJob, HDFS PVC, KFServing E2E Pipeline',
116 )
117 def e2e_pipeline():
118     # 데이터 준비 : Pylago를 이용해 Cuve에서 SQL로 데이터를 가져와 HDFS에 저장
119     transfer_task = transfer_op(
120         cuve_location=...,
121         cuve_sql=sql,
122         c3s_username=...,
123         hdfs_namespace=...,
124         hdfs_save_path=...,
125         transfer_queue=...,
126     ).apply(use_k8s_secret(secret_name='kerberos', k8s_secret_key_to_env={'keytab': 'KEYTAB'})) \
127         .apply(use_k8s_secret(secret_name='cuve', k8s_secret_key_to_env={'ticket': 'TICKET'}))
128     # Component caching 기능을 off
129     transfer_task.execution_options.caching_strategy.max_cache_staleness = "P0D"
```


3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

데이터 준비 (Cuve, CQuery) ->

데이터 전처리 (C3S Spark) ->

HDFS PVC 생성 (C3S HDFS) ->

모델 학습 (TFJob) ->

모델 서빙 (KFServing)

```
22 ##### PySpark #####
23 spark_code = """import argparse
24
25 from pyspark import SparkContext
26 from pyspark import SparkFiles
27 from pyspark.sql import Row
28 from pyspark.sql import SQLContext
29 from pyspark.sql.types import StructField, StructType, StringType, IntegerType, FloatType, ArrayType
30 from pyspark.sql.functions import udf
31
32 import requests
33
34
35 def refine(input_dir, output_dir):
36     [REDACTED]
37     [REDACTED]
38
39     sc = SparkContext()
```

...

```
131 # 데이터 전처리 : C3S 클러스터에서 Spark를 이용해 데이터를 전처리
132 sparkjob_task = sparkjob_op(
133     spark_code=spark_code,
134     requirements=requirements,
135     c3s_username='[REDACTED]',
136     hdfs_input=transfer_task.outputs["hdfs_path"],
137     hdfs_output='[REDACTED]',
138     queue='[REDACTED]'
139     ).apply(use_k8s_secret(secret_name='kerberos', k8s_secret_key_to_env={'keytab': 'KEYTAB'}))
140 sparkjob_task.execution_options.caching_strategy.max_cache_staleness = "P0D"
```

3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

데이터 준비 (Cuve, CQuery) ->

데이터 전처리 (C3S Spark) ->

HDFS PVC 생성 (C3S HDFS) ->

모델 학습 (TFJob) ->

모델 서빙 (KFServing)

```
142 # HDFS PVC 생성 : C3S HDFS에 저장된 데이터 사용을 위해 Alluxio PVC 생성
143 data_task = kfp.dsl.VolumeOp(
144     name="Create Alluxio PVC",
145     resource_name="tf-mnist-hdfs",
146     size="1Gi",
147     storage_class="alluxio",
148     modes=["ReadWriteMany"],
149     annotations={
150         "hdfs/namespace": sparkjob_task.outputs["hdfs_namespace"],
151         "hdfs/path": sparkjob_task.outputs["hdfs_short_path"],
152         "hdfs/mkdirIfNotExists": "true"
153     },
154     ).after(sparkjob_task)
```


3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

데이터 준비 (Cuve, CQuery) ->

데이터 전처리 (C3S Spark) ->

HDFS PVC 생성 (C3S HDFS) ->

모델 학습 (TFJob) ->

모델 서빙 (KFServing)

```
48 def main():
49     train_data, test_data = load_data()
50
51     # Construct a model
52     X = tf.placeholder(tf.float32, [None, 784])
53     Y = tf.placeholder(tf.float32, [None, 10])
54
55     logits = convnet(X)
56
57     logits_test = convnet(X, is_training=False, reuse=True)
58     prediction = tf.nn.softmax(logits_test)
```

...

Tensorflow
모델 학습 코드

TFJob 배포 코드

```
77 ##### TFJob #####
78 tfjob_worker_yaml="""
79 replicas: 1
80 restartPolicy: Never
81 template:
82   metadata:
83     annotations:
84       sidecar.istio.io/inject: "false"
85   spec:
86     containers:
87       - name: tensorflow
```

...

```
156 # 모델 학습 : TFJob을 이용해 모델 학습을 진행
157 training_task = tfjob_launcher_op(
158     name="mnist",
159     namespace="wangyuhan",
160     ttl_seconds_after_finished=-1,
161     worker_spec=tfjob_worker_yaml.format(data_task.outputs['name']),
162     chief_spec={},
163     tfjob_timeout_minutes=60,
164     delete_finished_tfjob=True
165 ).after(data_task)
166 training_task.execution_options.caching_strategy.max_cache_staleness = "P0D"
```

3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

데이터 준비 (Cuve, CQuery) ->

데이터 전처리 (C3S Spark) ->

HDFS PVC 생성 (C3S HDFS) ->

모델 학습 (TFJob) ->

모델 서빙 (KFServing)

```
168     # 모델 서빙 : KFServing을 이용해 Tensorflow 모델을 Tensorflow Serving으로 서빙 진행
169     serving_task = kfserving_op(
170         action="apply", # apply = create or update
171         namespace="wangyuhan",
172         model_name="tf-mnist",
173         framework="tensorflow",
174         model_uri=f"pvc://{data_task.outputs['name']}/trained_model",
175     ).after(training_task)
176     serving_task.execution_options.caching_strategy.max_cache_staleness = "P0D"
```

전체 190줄의 코드로 구성 가능

3.5 사내 플랫폼을 연계한 ML 파이프라인 구성

데이터 준비 (Cuve, CQuery) ->

데이터 전처리 (C3S Spark) ->

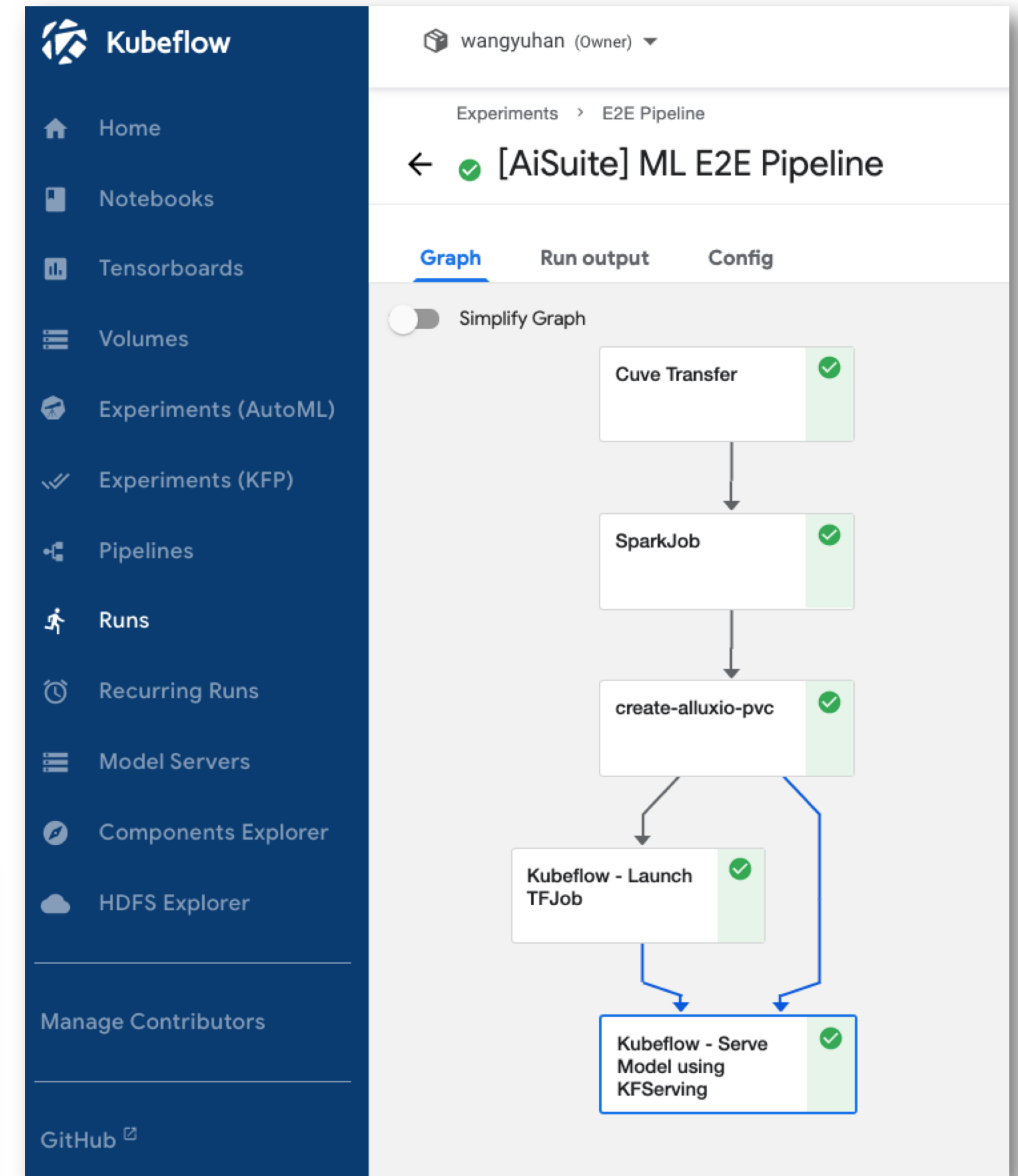
HDFS PVC 생성 (C3S HDFS) ->

모델 학습 (TFJob) ->

모델 서빙 (KFServing)

```
→ tf-mnist git:(master) x curl -X POST -H "Accept: application/json" http://tf-mnist.wan  
gyuhan. /v1/models/tf-mnist:predict -d @input.json  
{  
  "predictions": [[0.0447704382, 0.332736582, 0.0966353193, 0.055440288, 0.102505483, 0  
.0576355532, 0.0804797933, 0.0749690384, 0.0740943104, 0.08073318]  
]  
}  
→ tf-mnist git:(master) x |
```

쿼리 요청 결과



3.6 그 외 파이프라인 예시

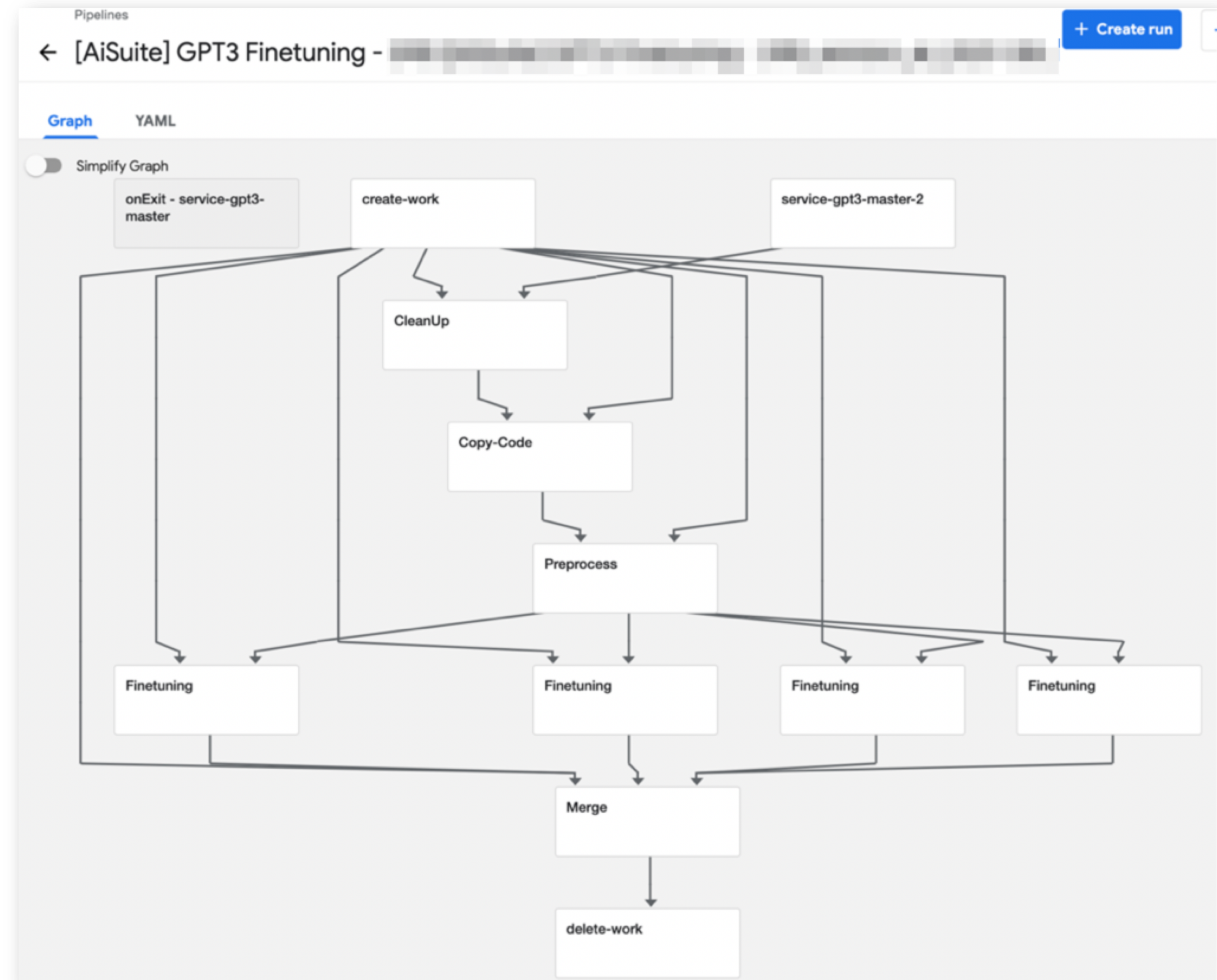
GPT3 파인튜닝

HDFS PVC 생성 ->

데이터 전처리 ->

모델 학습 (분산 학습) ->

학습된 모델 병합

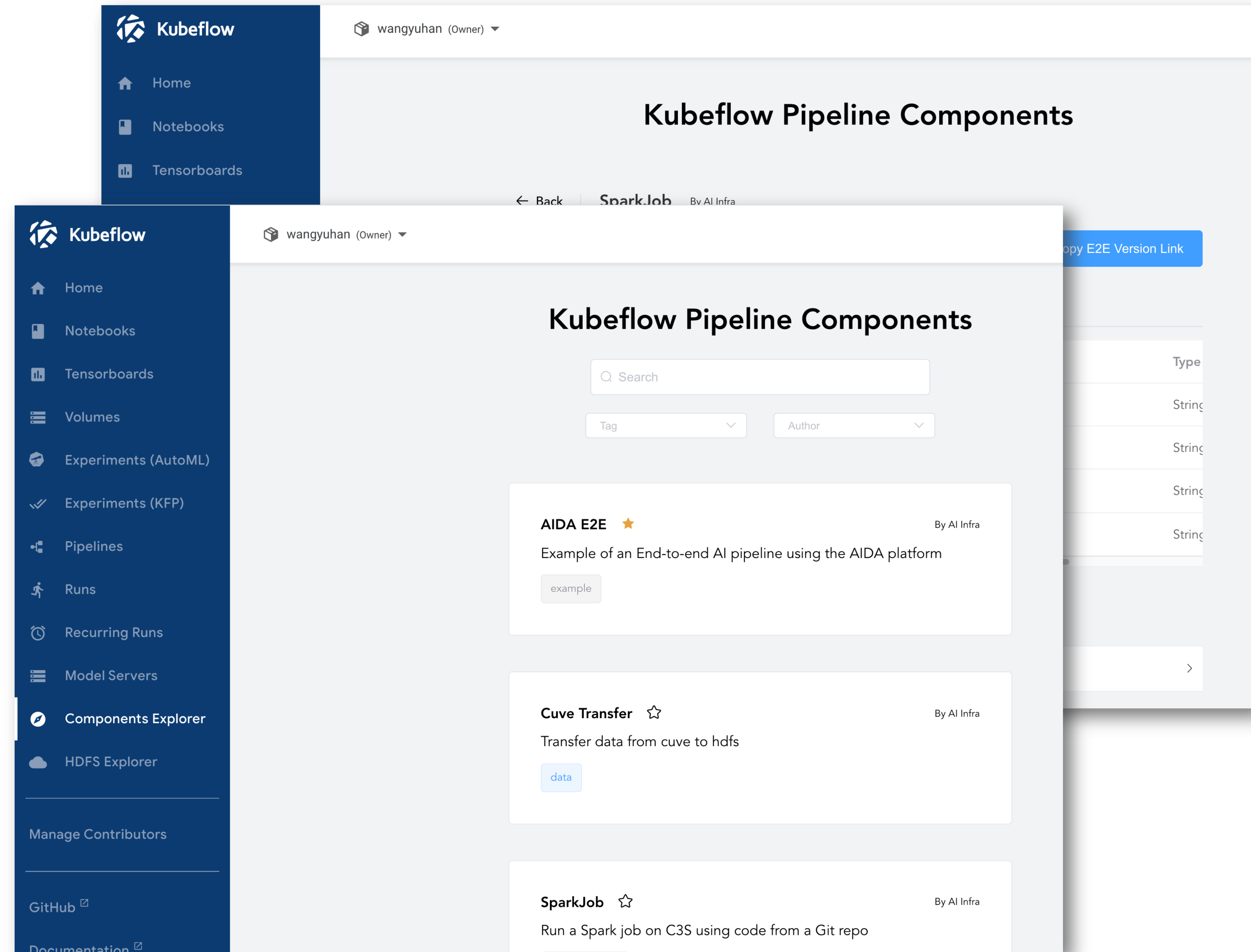


3.7 파이프라인 공유

Components Explorer

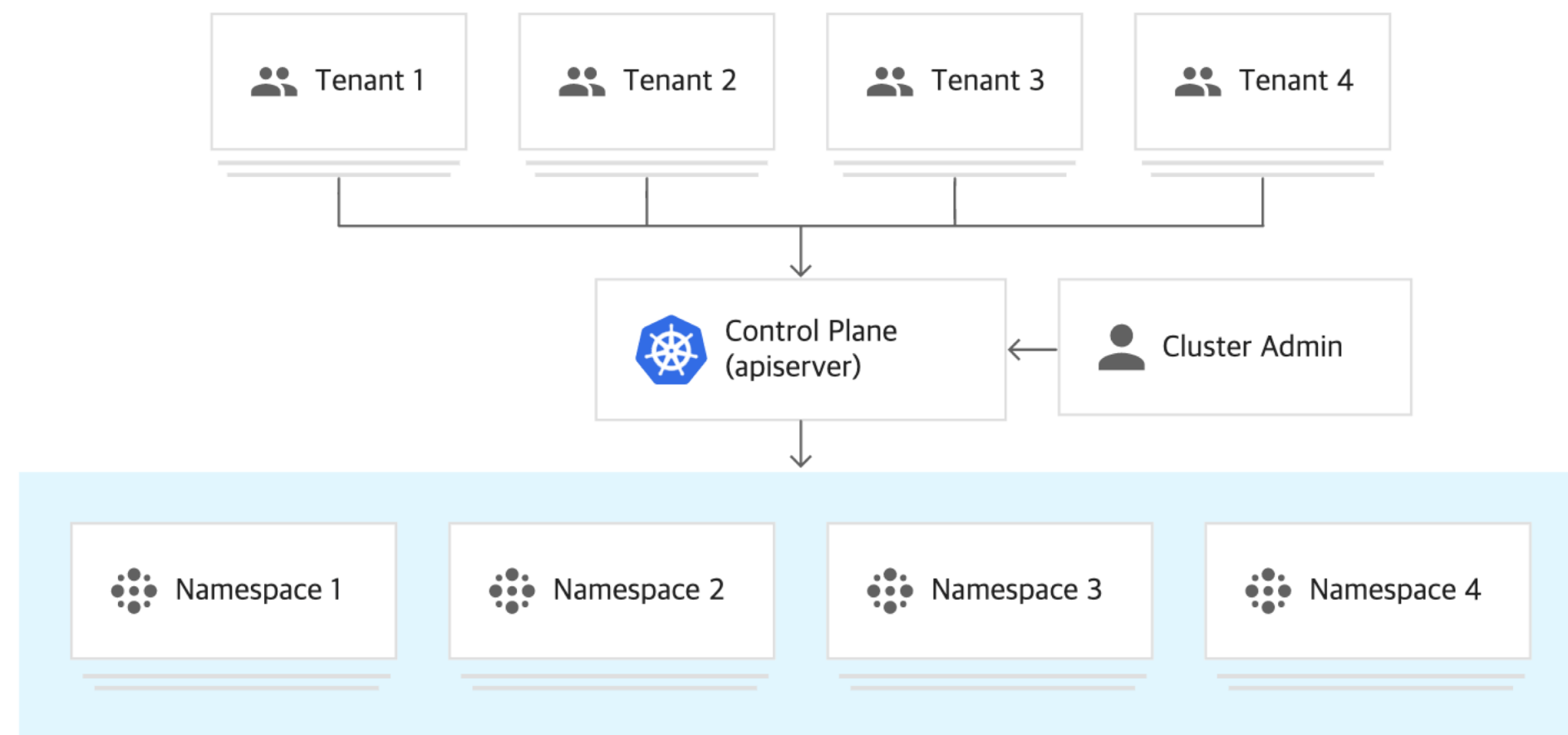
- 파이프라인 모듈 및 예시들을
Kubeflow 대쉬보드를 통해
공유할 수 있는 공간

=> 사내 개발자들이 새로운
모듈을 개발하고 공유하도록
적극적으로 지원

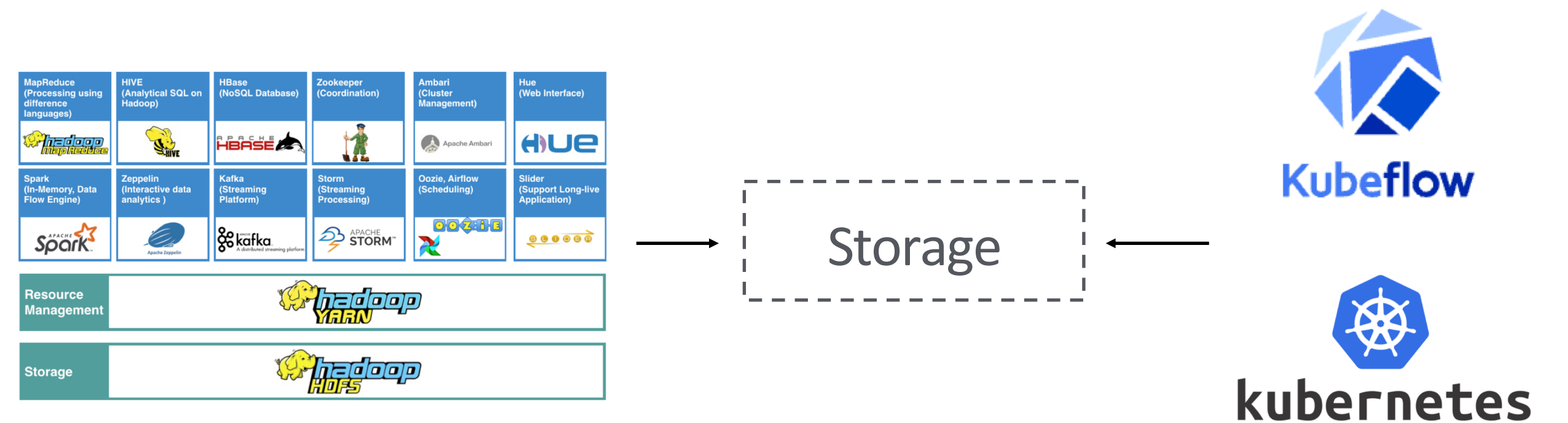


4. Multi-Tenant Kubeflow

4. 플랫폼 구축 시 요구사항



1. Multi-Tenancy



2. ML을 위한 Storage (5장에서 설명)

* Multi-Tenant Cluster: A multi-tenant cluster is shared by multiple users and/or workloads which are referred to as "tenants".

4.1 Multi-Tenancy를 위해 필요한 기능들

인증

액세스 제어

작업 공간 분리

리소스 할당 제한

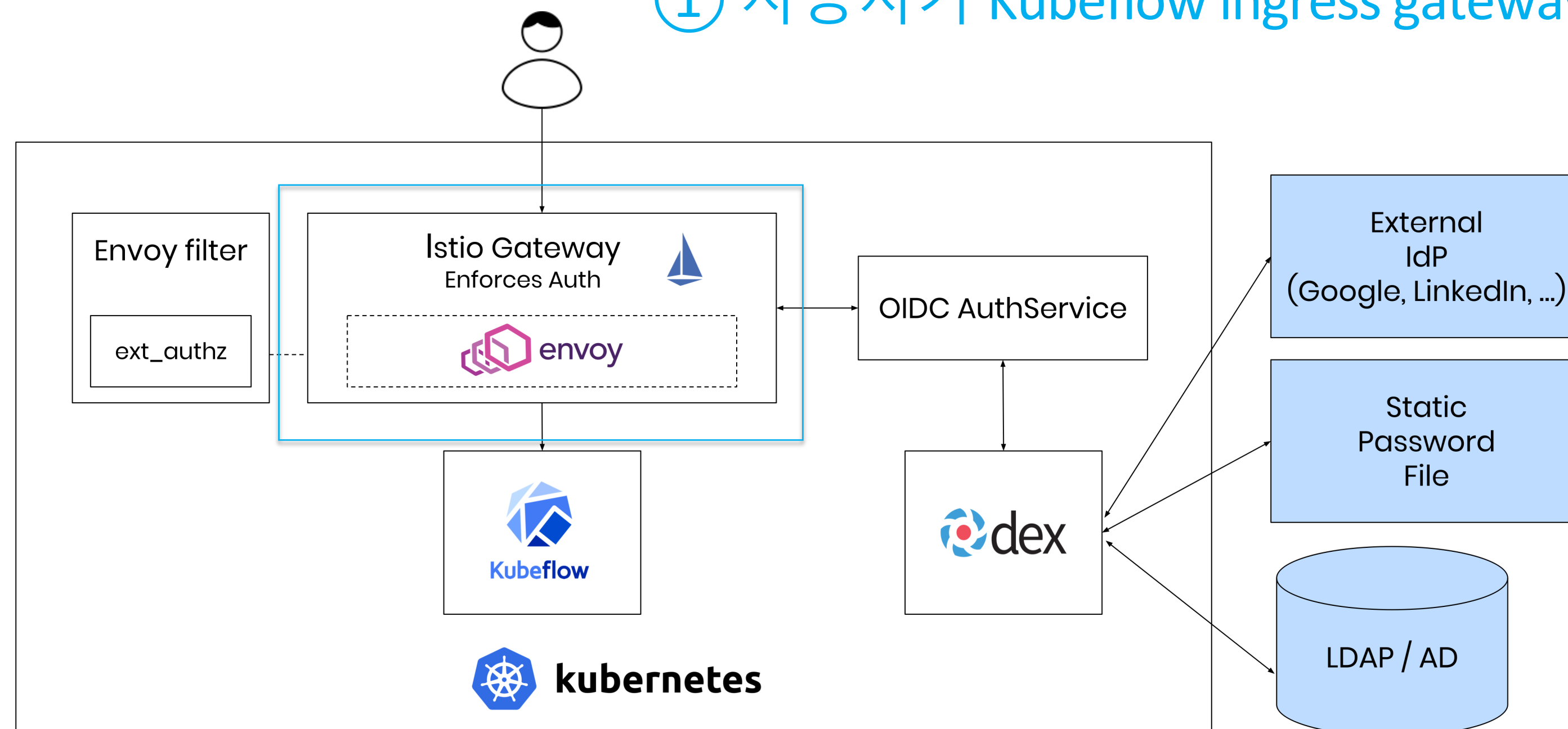
4.1 Multi-Tenancy를 위해 필요한 기능들



4.2.1 KubeFlow의 Multi-Tenancy 지원 – 인증

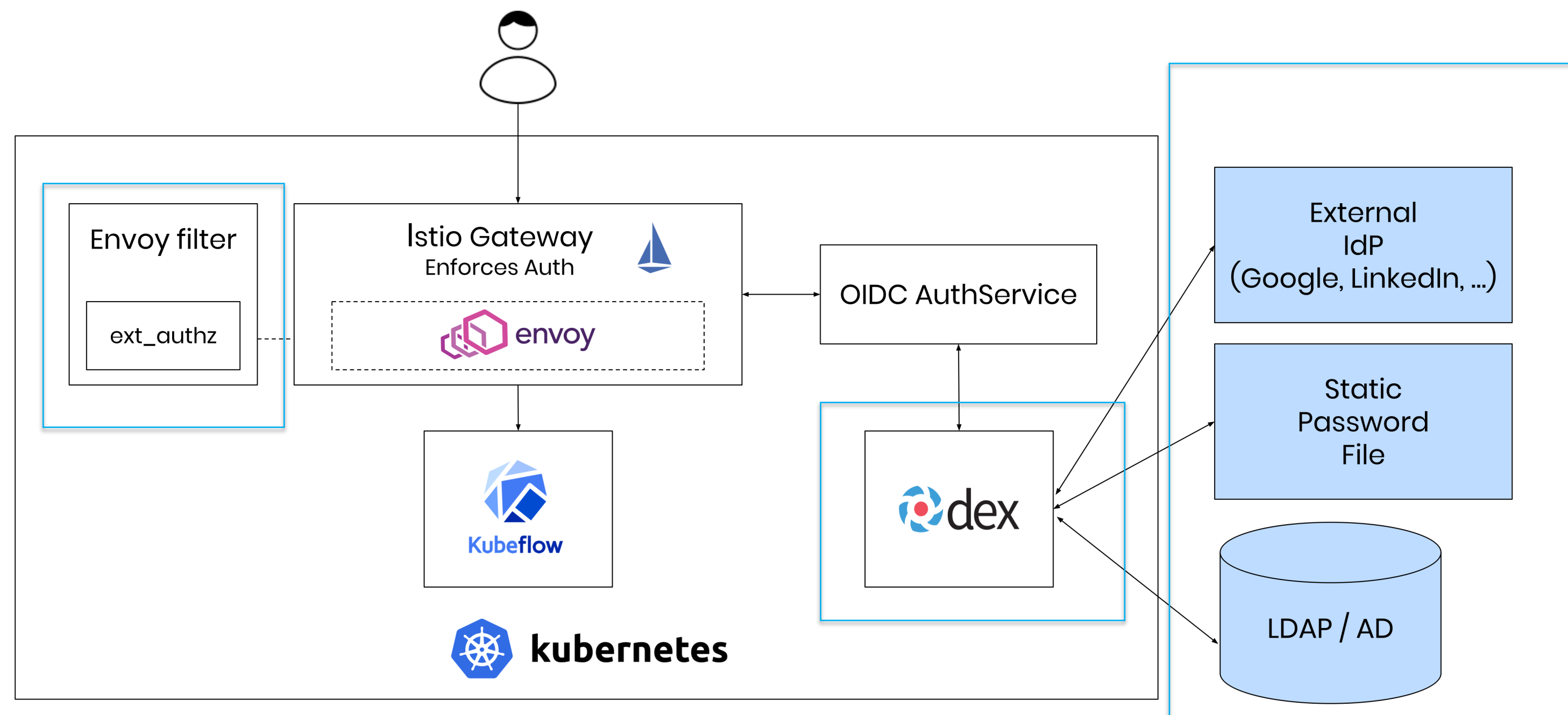
KubeFlow 대시보드 인증 절차

① 사용자가 KubeFlow ingress gateway 통해 접근



4.2.1 KubeFlow의 Multi-Tenancy 지원 – 인증

KubeFlow 대시보드 인증 절차



Pluggable한 구조로
다양한 인증 방식을 도입 가능

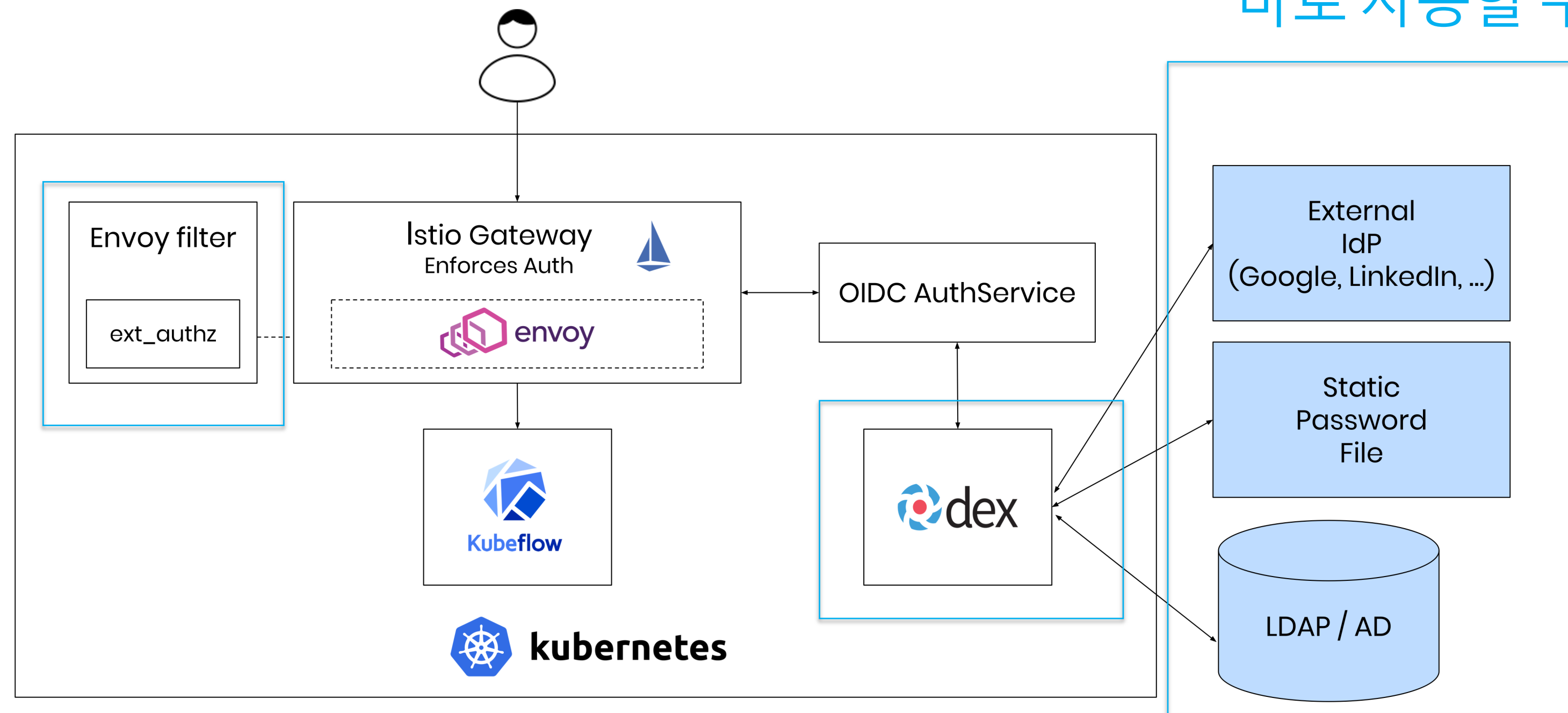
② Istio EnvoyFilter가 적용되어 dex에 인증 요청

4.2.1 KubeFlow의 Multi-Tenancy 지원 – 인증

KubeFlow 대시보드 인증 절차

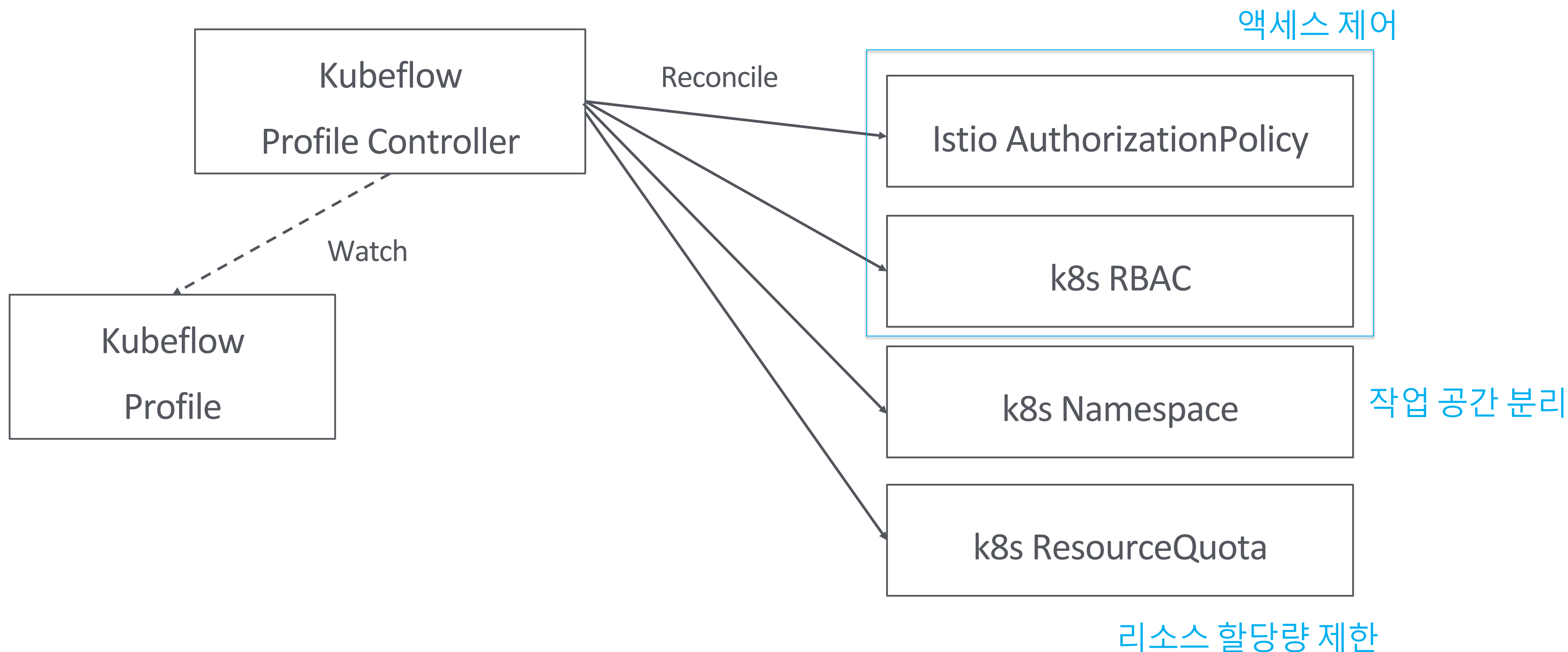
③ 기존 C3S, C3DL 사용자가

바로 사용할 수 있도록 C3S LDAP에 인증 위임



4.2.2 KubeFlow의 Multi-Tenancy 지원

KubeFlow Profile을 통한 multi-tenancy 지원



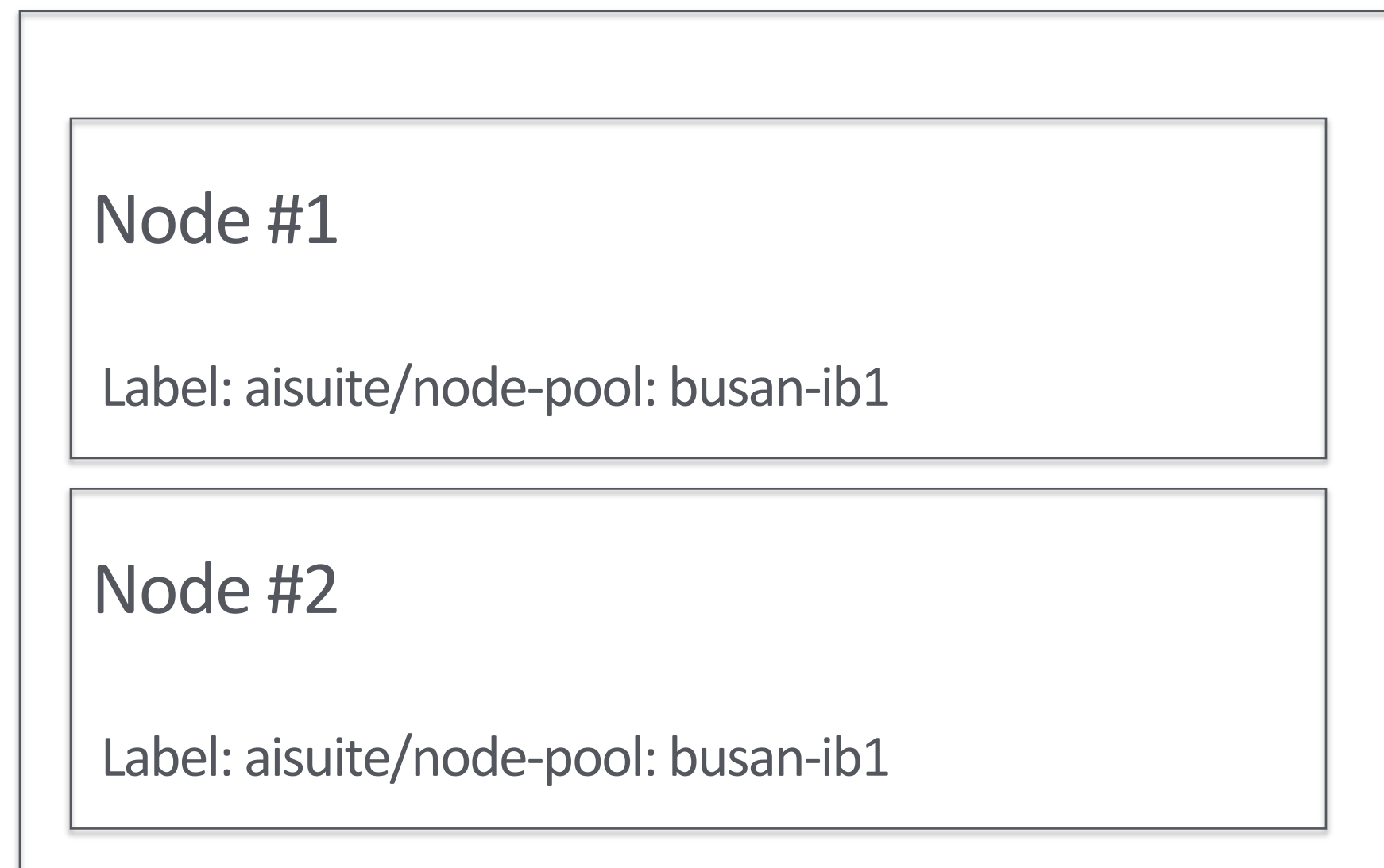
4.3 우리 환경에 필요한 요구사항

전용 node pool 지원

- 중요한 서비스의 경우 다른 tenant에 영향 받지 않는 독립된 node pool 지원 필요
- 소속 혹은 그룹별로 전용 node를 쓰고자 하는 경우 지원 필요
- 분산 학습 시 InfiniBand topology 고려해 pod 스케줄링 필요
- 허용된 사용자만 node pool에 pod을 배포하도록 제한 필요

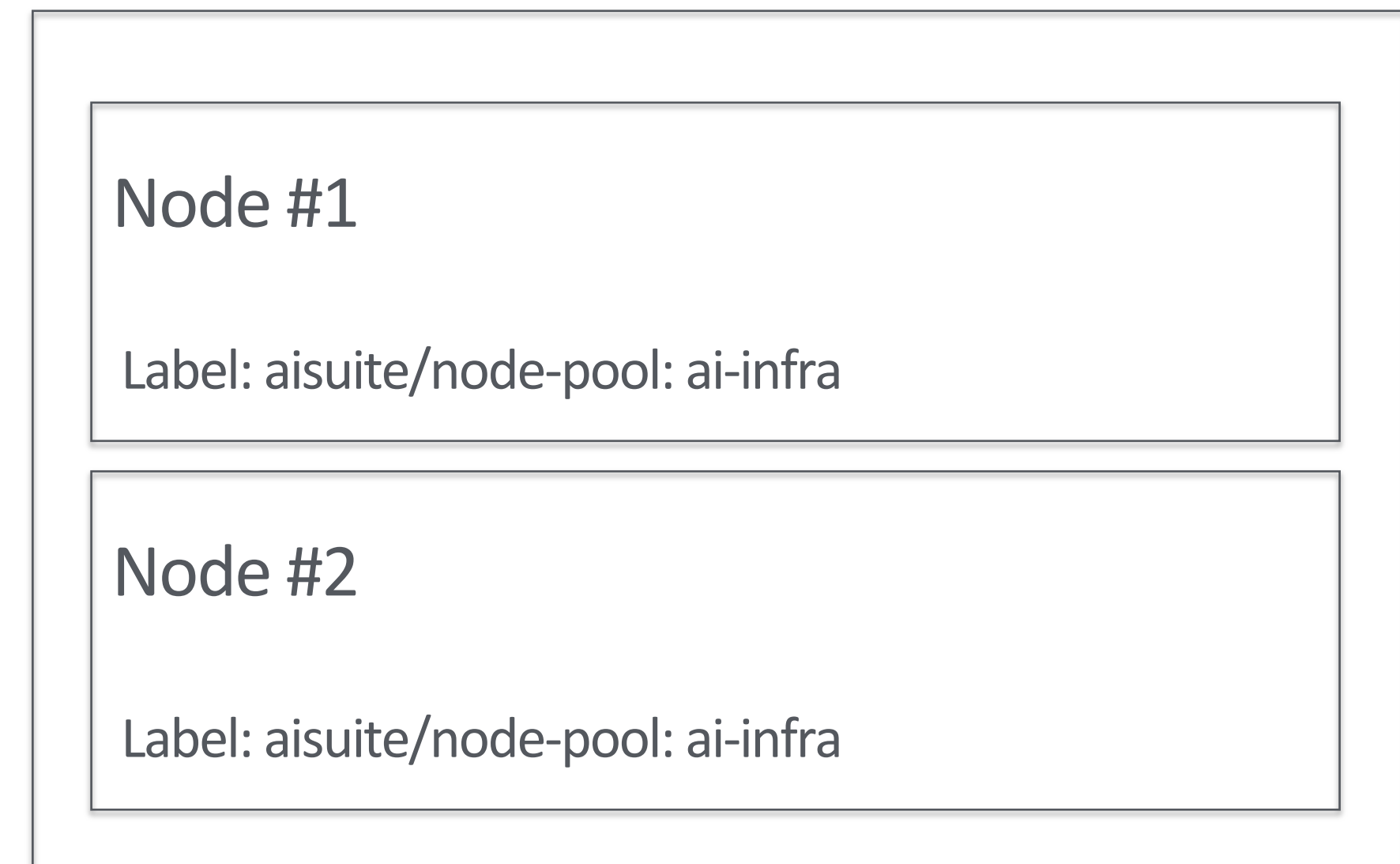
4.4 전용 Node pool 지원 – (1) Node pool 구분

Node pool 설정: Kubernetes node label을 활용해 node pool 구분



Node pool: busan-ib1

부산IDC / 1번 InfiniBand switch에 묶인 노드

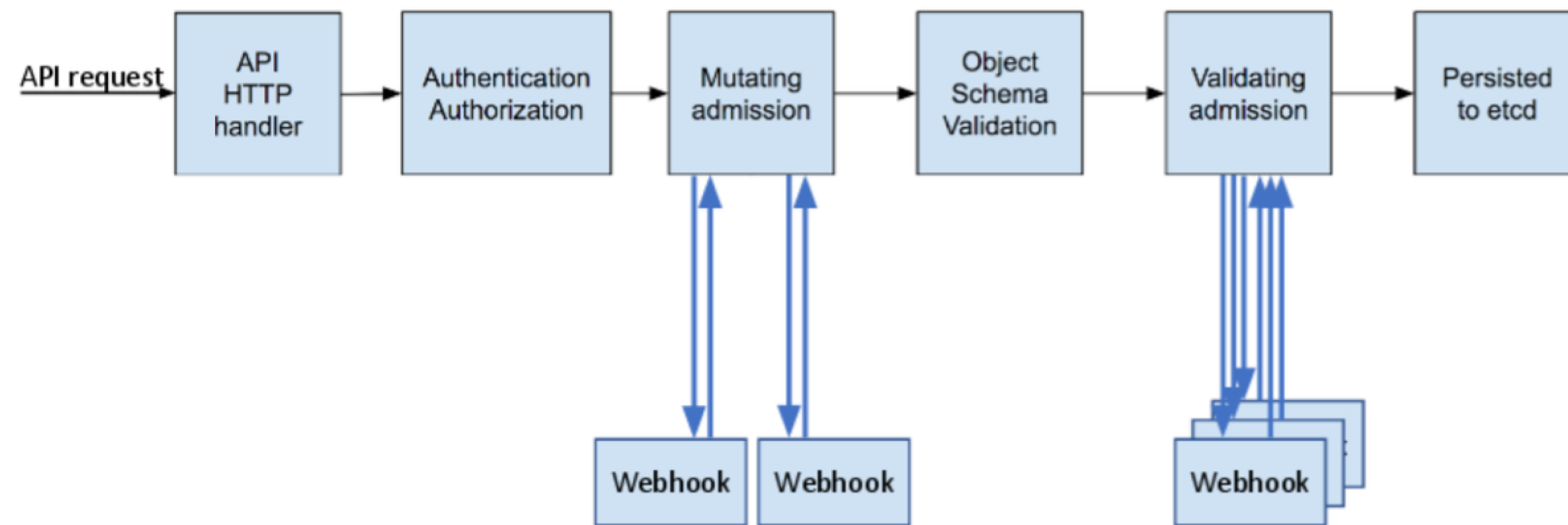


Node pool: ai-infra

AI Infra 팀의 전용 노드

4.4 전용 Node pool 지원 - (2) 스케줄링

Node pool에 pod 스케줄링: Kubernetes admission webhook 활용



Kubernetes admission webhook

Kubernetes API 서버로 가는 요청을 중간에 가로채는 webhook을 만들어 해당 요청을 **변환** (mutating admission) 하거나 **검증** (validating admission) 하는 것이 가능.

4.4 전용 Node pool 지원 – (3) Preemptible pod

Cluster Utilization: Kubernetes PriorityClass를 활용한 preemptible pod 지원

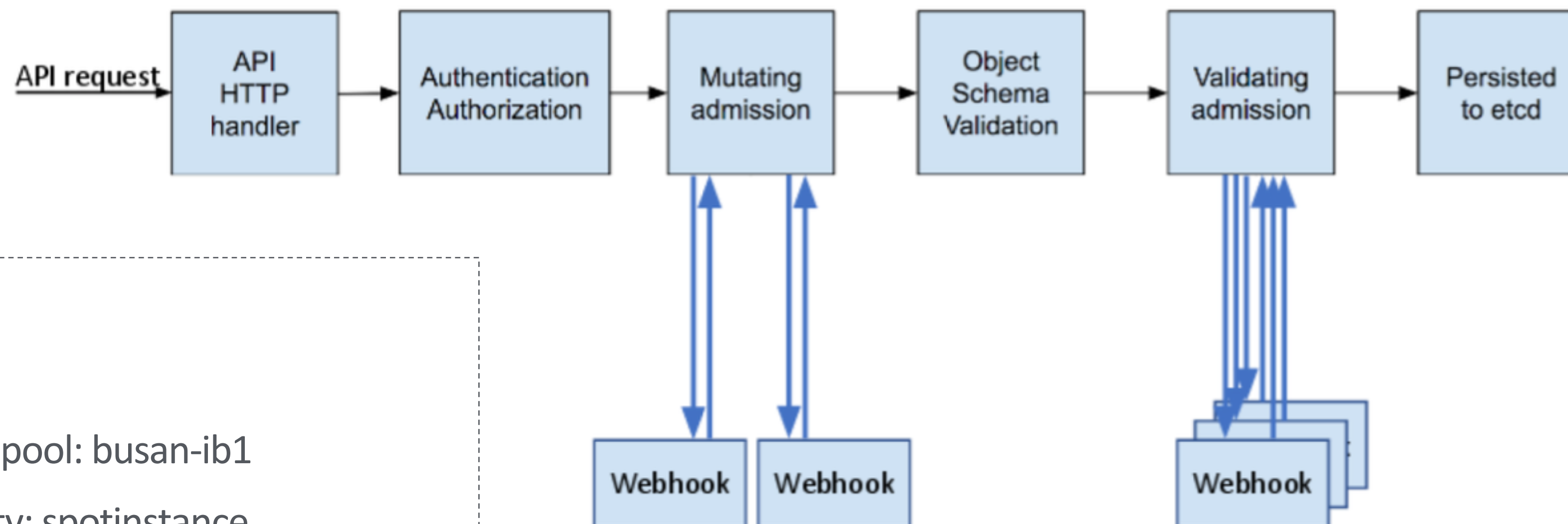
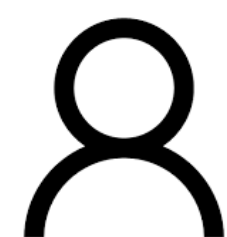
```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: spotinstance
value: 900000
preemptionPolicy: Never
globalDefault: false
description: Spotinstance can be terminated at anytime
```

우선순위가 높은 pod 스케줄링 불가 시

→ 낮은 우선순위를 가진 pod이 **preemption**됨

Kubernetes PriorityClass

4.4.1 전용 node pool 스케줄링 구현



1. Pod 생성 요청

metadata:

annotations:

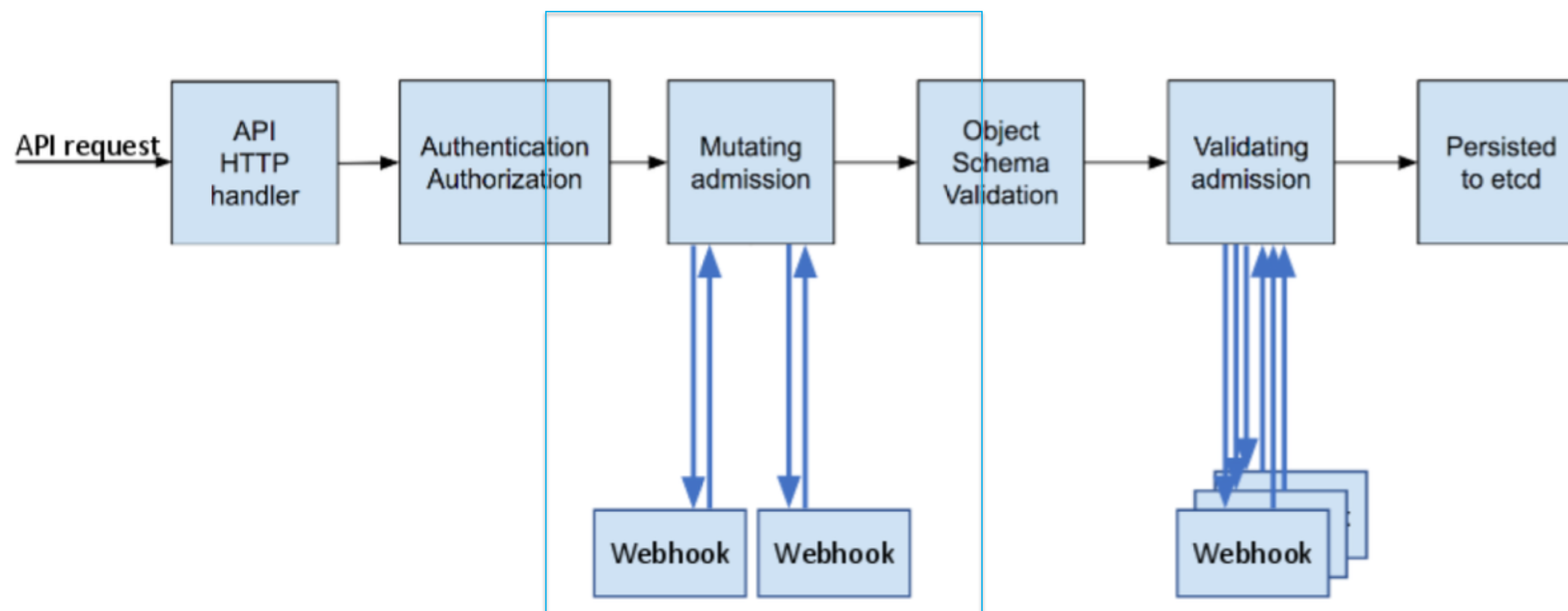
aisuite.navercorp.com/node-pool: busan-ib1

aisuite.navercorp.com/priority: spotinstance

4.4.1 전용 node pool 스케줄링 구현



1. Pod 생성 요청



2. Mutation (scheduling)

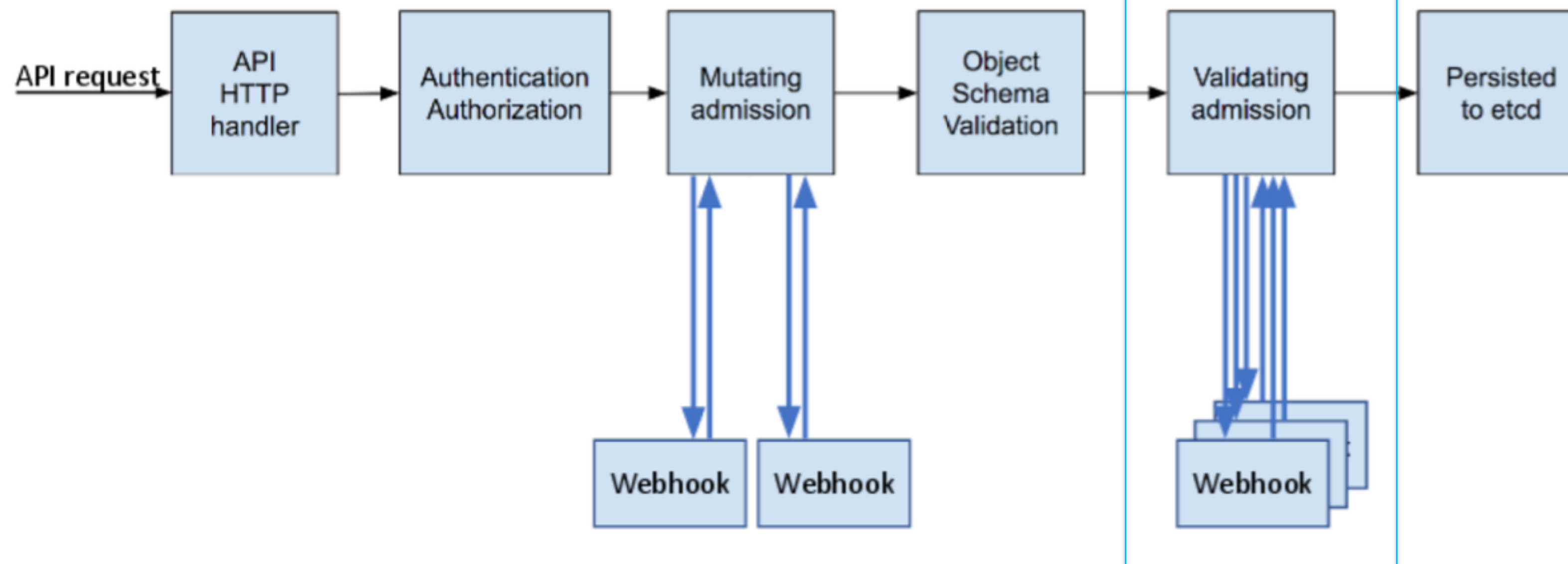
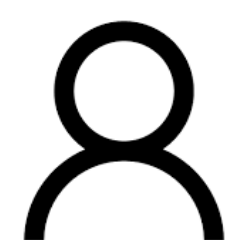
- Node pool 설정 X → 공용 node pool 에 스케줄링
- Preemptible pod 요청 → 가장 여유 있는 node pool에 스케줄링

4.4.1 전용 node pool 스케줄링 구현

3. Validation

허용된 node pool에 pod을 요청했는지 검증

1. Pod 생성 요청



2. Mutation (scheduling)

4.5 운영을 편리하게: 선언적으로 사용자 관련 정책 관리하기



상황에 따라 사용자마다 Kubeflow Profile과 별도로 관리해야 하는 설정들...

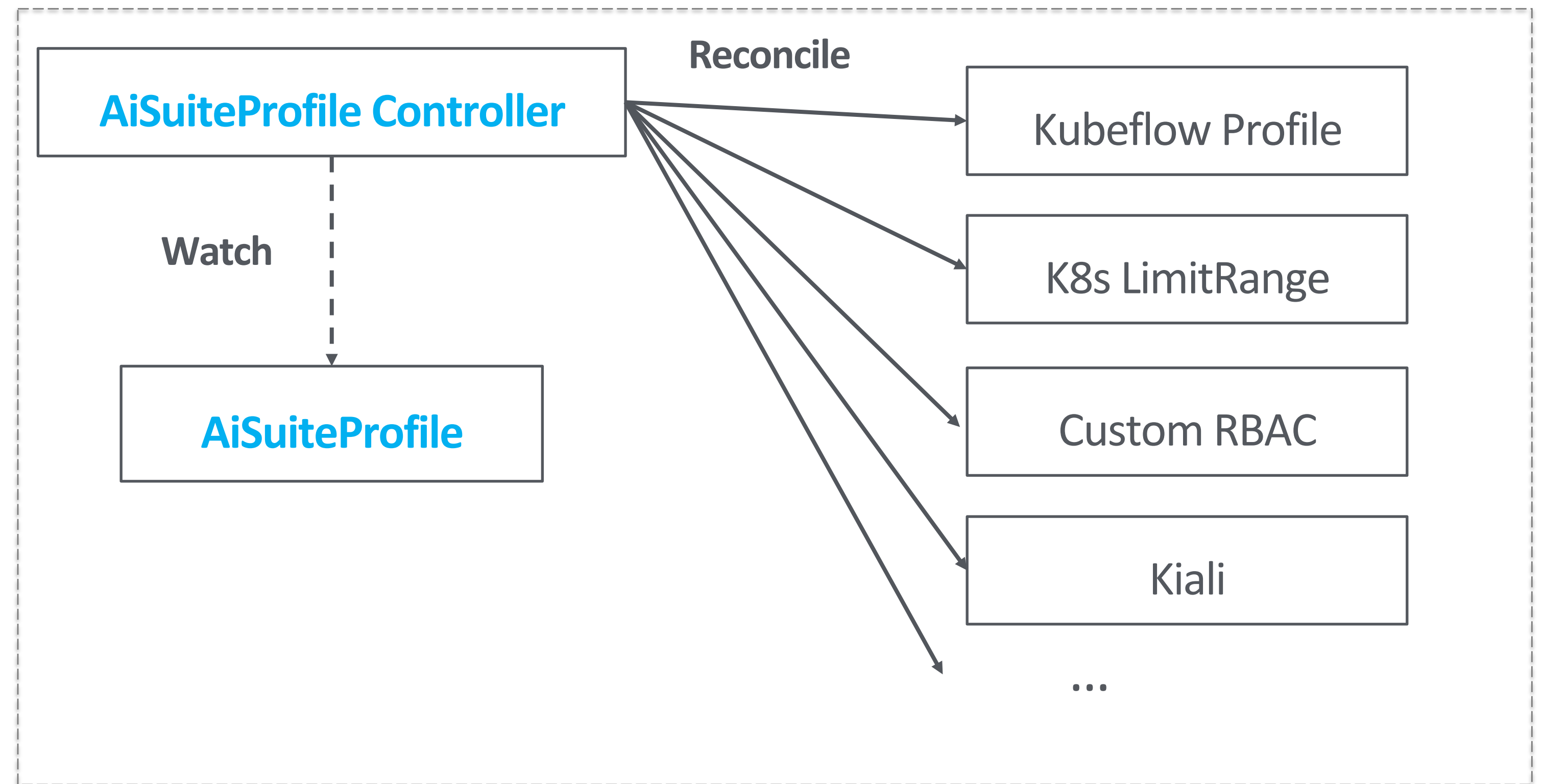
- 전용 node pool 관련 설정
 - Kubernetes LimitRange 설정
 - KFServing 사용 편의를 위한 인증 우회 설정
 - Kubeflow notebook 에서 Pipeline SDK 사용 편의를 위한 인증 우회 설정
 - 사용자 별로 요청한 CRD에 대한 추가적인 권한 설정
 - 운영자 입장에서 이러한 요구사항들을 ad-hoc 하게 반영하기는 어려움
- 선언적으로 사용자 관련 정책 및 리소스들을 배포 및 관리 필요

4.5.1 사용자의 모든 정보를 담은 리소스: AiSuiteProfile

Kubeflow Profile을 확장한 custom resource 도입: AiSuiteProfile

```
apiVersion: multitenancy.aisuite.navercorp.com/v1alpha1
kind: AiSuiteProfile
metadata:
  name: jwpark
spec:
  zone: search
  exclusiveZones:
    - ai-infra
  limitRange:
    limits:
      - default:
          cpu: 500m
          ephemeral-storage: 20G
          memory: 1Gi
      max:
          cpu: "60"
          ephemeral-storage: 400G
          memory: 1Ti
    type: Container
  resourceQuota:
    hard:
      persistentvolumeclaims: "100"
      alluxio.storageclass.storage.k8s.io/persistentvolumeclaims: "20"
  kiali:
    enabled: false
```

AiSuiteProfile 예시



AiSuiteProfile Controller를 통한 k8s object 동기화

4.5.2 AiSuiteProfile Controller 구현: Metacontroller

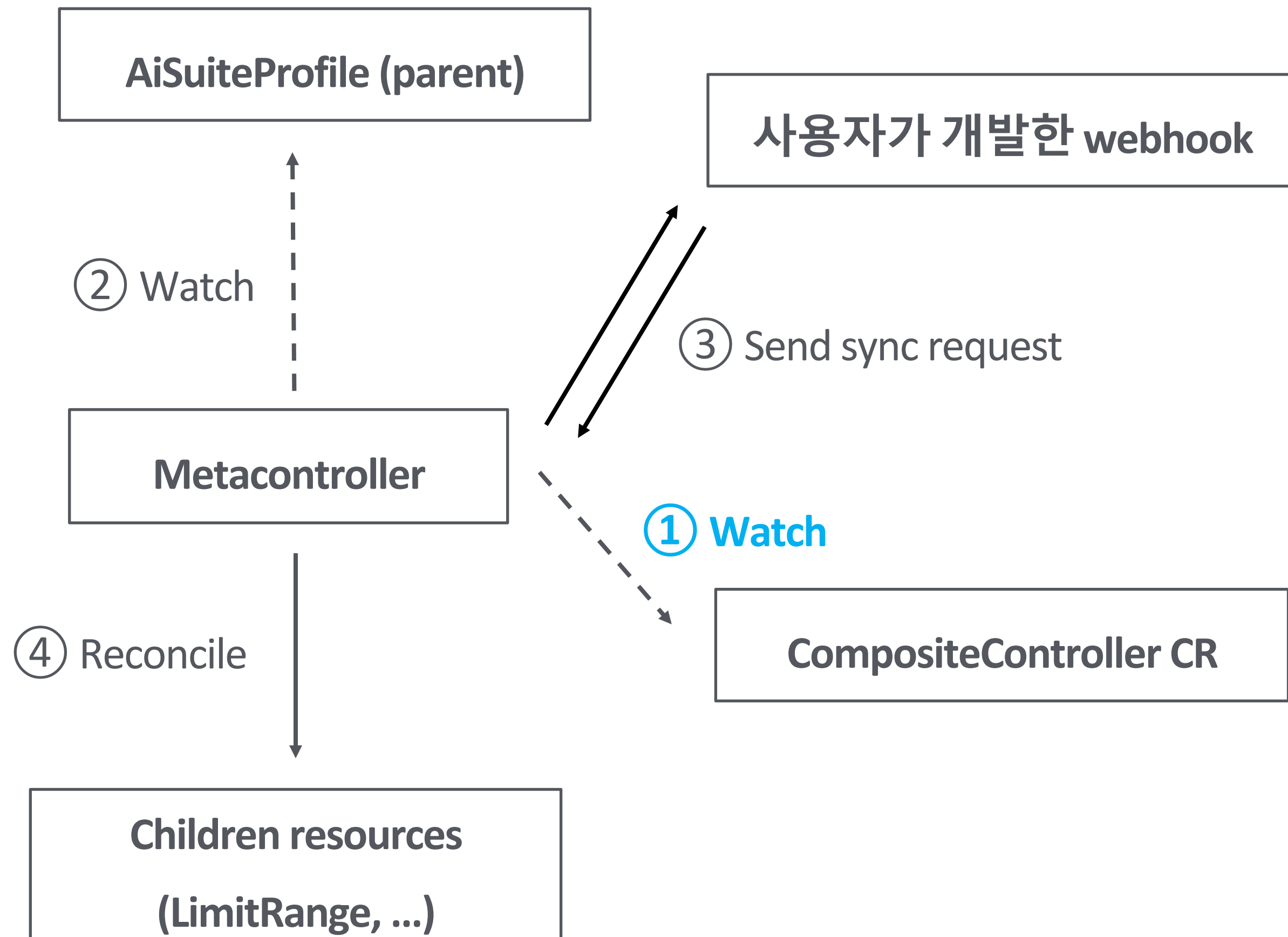
Metacontroller

- Custom Kubernetes controller를 쉽게 개발할 수 있도록 도와주는 툴
- Kubernetes controller 개발 시 **공통된 패턴을 추상화**
- 개발자가 webhook으로 controller의 동작을 작성하면 reconciliation 등은 metacontroller가 대신 수행

Metacontroller에서 추상화하여 제공하는 controller (Kubernetes custom resource로 제공)

- CompositeController: 사용자가 **선언한 parent 리소스에 맞춰 children 리소스들을 동기화**
- DecoratorController: 특정 리소스가 생성 / 삭제 될 때 해당 리소스를 annotate 하는 등 decoration을 해주는 controller

4.5.2 AiSuiteProfile Controller 구현: Metacontroller



```

apiVersion: metacontroller.k8s.io/v1alpha1
kind: CompositeController
metadata:
  name: aisuite-controller
spec:
  generateSelector: true
  resyncPeriodSeconds: 20
  parentResource:
    apiVersion: multitenancy.aisuite.navercorp.com/v1alpha1
    resource: aisuiteprofiles
  childResources:
    - apiVersion: kubeflow.org/v1
      resource: profiles
      updateStrategy:
        method: InPlace
    - apiVersion: security.istio.io/v1beta1
      resource: authorizationpolicies
      updateStrategy:
        method: InPlace
  hooks:
    sync:
      webhook:
        url: http://aisuite-controller.kubeflow/v1/sync/aisuite-profile
    
```

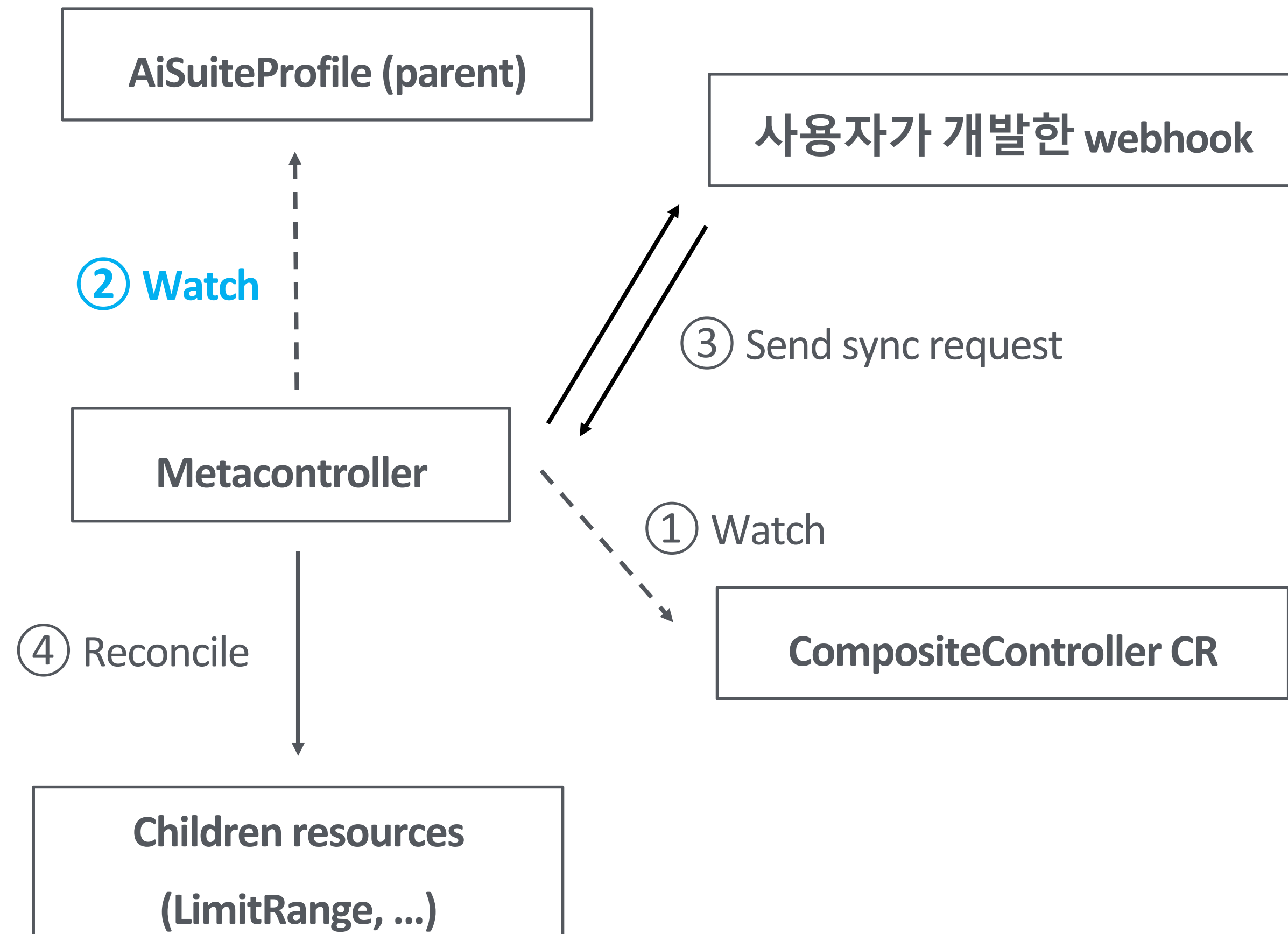
기준이 되는 parent 리소스 (Parent resource to be synchronized)

동기화될 children 리소스 종류 / 업데이트 전략 (Child resource types / update strategy)

Webhook URL

CompositeController 예시

4.5.2 AiSuiteProfile Controller 구현: Metacontroller



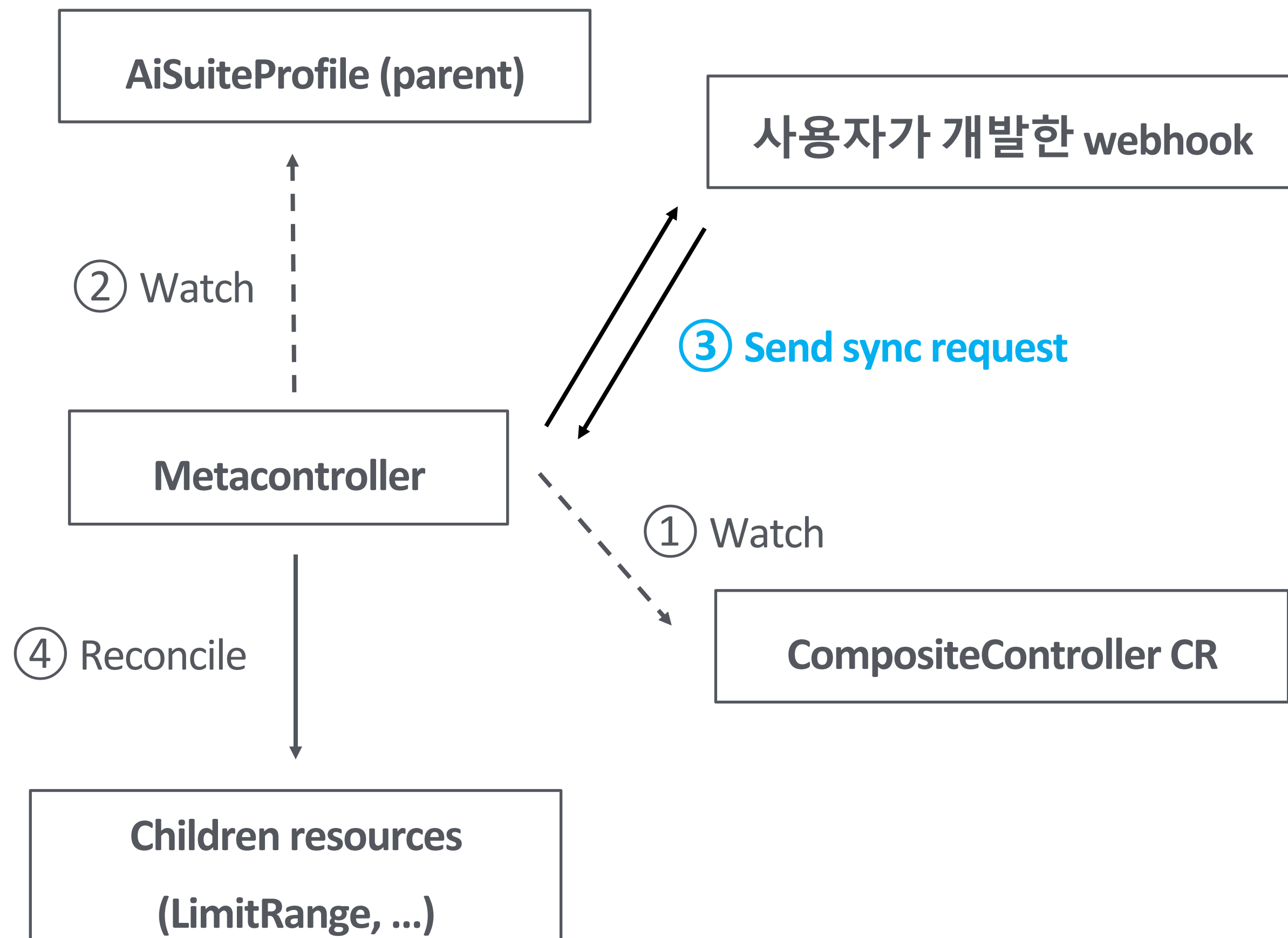
```

apiVersion: multitenancy.aisuite.navercorp.com/v1alpha1
kind: AiSuiteProfile
metadata:
  name: jwpark
spec:
  zone: search
  exclusiveZones:
    - ai-infra
  limitRange:
    limits:
      - default:
          cpu: 500m
          ephemeral-storage: 20G
          memory: 1Gi
        max:
          cpu: "60"
          ephemeral-storage: 400G
          memory: 1Ti
        type: Container
  resourceQuota:
    hard:
      persistentvolumeclaims: "100"
      alluxio.storageclass.storage.k8s.io/persistentvolumeclaims: "20"
  kiali:
    enabled: false
  
```

사용자 설정에 대한 정보

AiSuiteProfile 예시

4.5.2 AiSuiteProfile Controller 구현: Metacontroller



Webhook 예시

```

@router.post("/aisuite-profile")
async def sync_aisuite_profile(request: SyncRequest):
    namespace = request.parent.get("metadata", {}).get("name")
    limit_range_spec = request.parent.get("spec", {}).get("limitRange", {})
    kiali_spec = request.parent.get("spec", {}).get("kiali", {})
    exclusive_zones = request.parent.get("spec", {}).get("exclusiveZones", [])

    ...

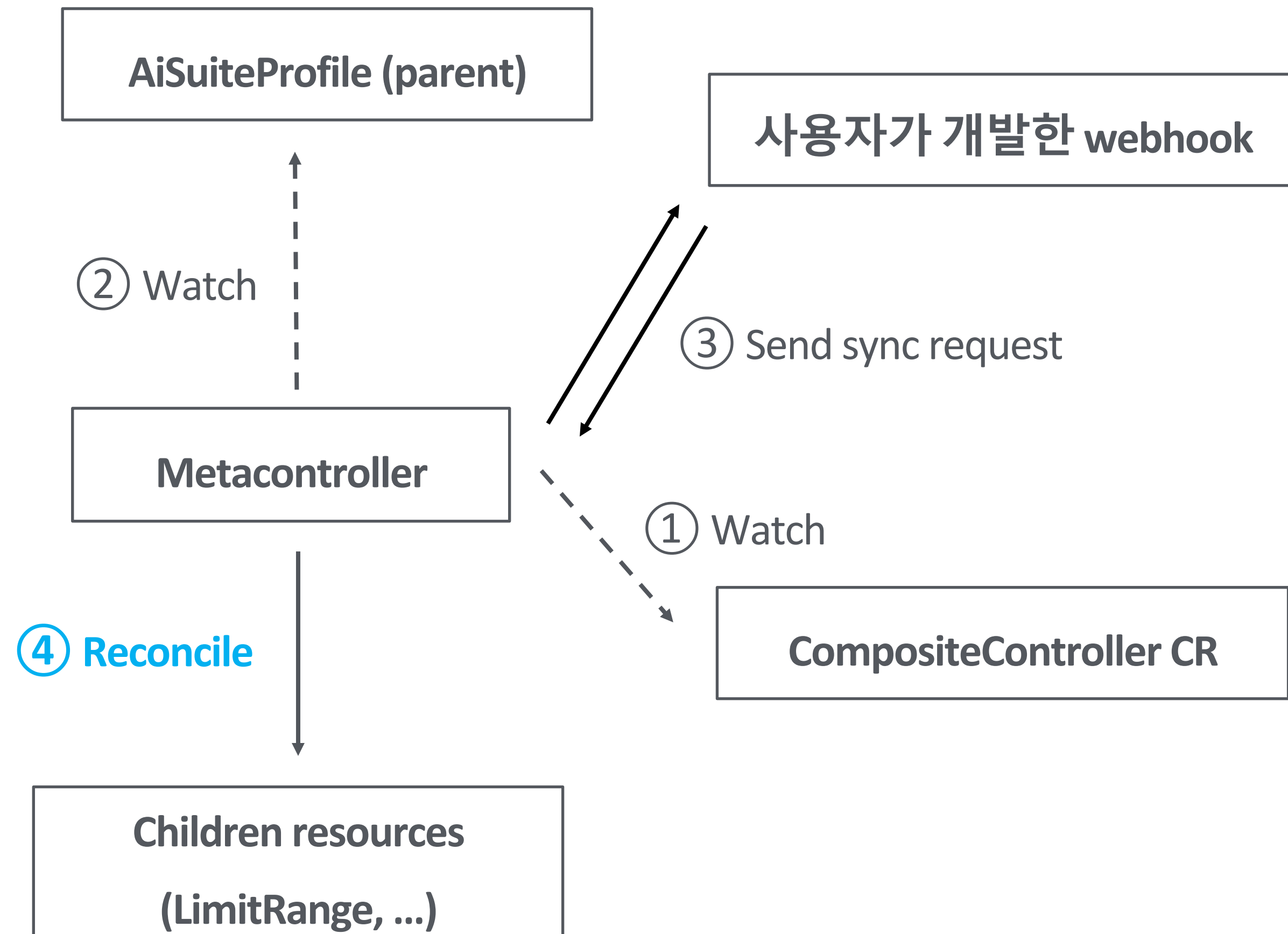
    desired_resources = []
    desired_resources.append(_profile(namespace, resource_quota))

    ...

    return SyncResponse(status=current_status, children=desired_resources)
  
```

- Request: Parent 리소스 (AiSuiteProfile)
- Response: 동기화할 children 리소스

4.5.2 AiSuiteProfile Controller 구현: Metacontroller



```

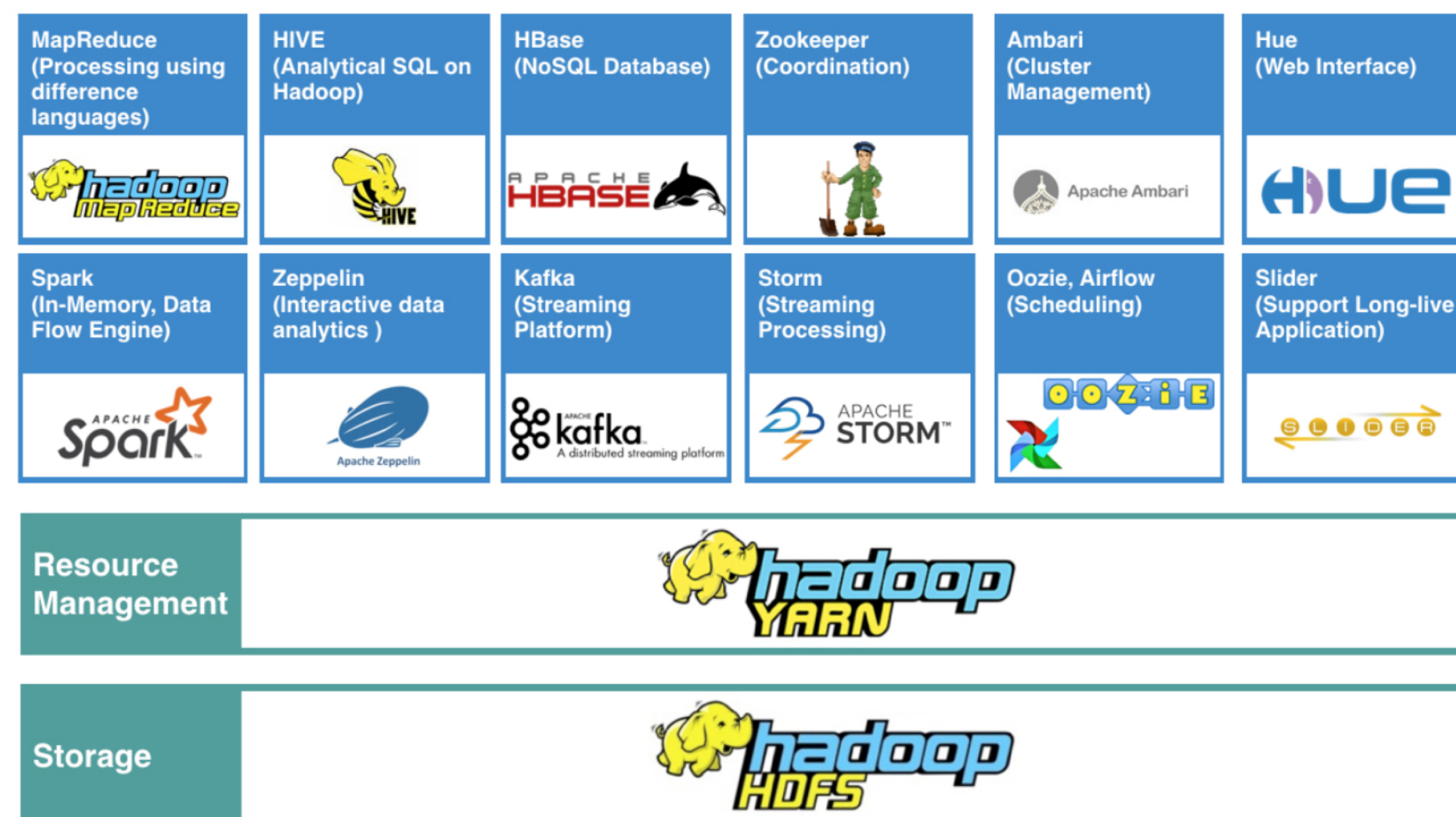
apiVersion: metacontroller.k8s.io/v1alpha1
kind: CompositeController
metadata:
  name: aisuite-controller
spec:
  generateSelector: true
  resyncPeriodSeconds: 20
  parentResource:
    apiVersion: multitenancy.aisuite.navercorp.com/v1alpha1
    resource: aisuiteprofiles
  childResources:
    - apiVersion: kubeflow.org/v1
      resource: profiles
      updateStrategy:
        method: InPlace
    - apiVersion: security.istio.io/v1beta1
      resource: authorizationpolicies
      updateStrategy:
        method: InPlace
  hooks:
    sync:
      webhook:
        url: http://aisuite-controller.kubeflow/v1/sync/aisuite-profile
  
```

동기화될 children 리소스
종류 / 업데이트 전략

CompositeController 예시

5. HDFS를 k8s 볼륨으로 사용하기 위한 Alluxio 연동

5.1 ML 학습을 위한 Storage



데이터 처리 / 보관

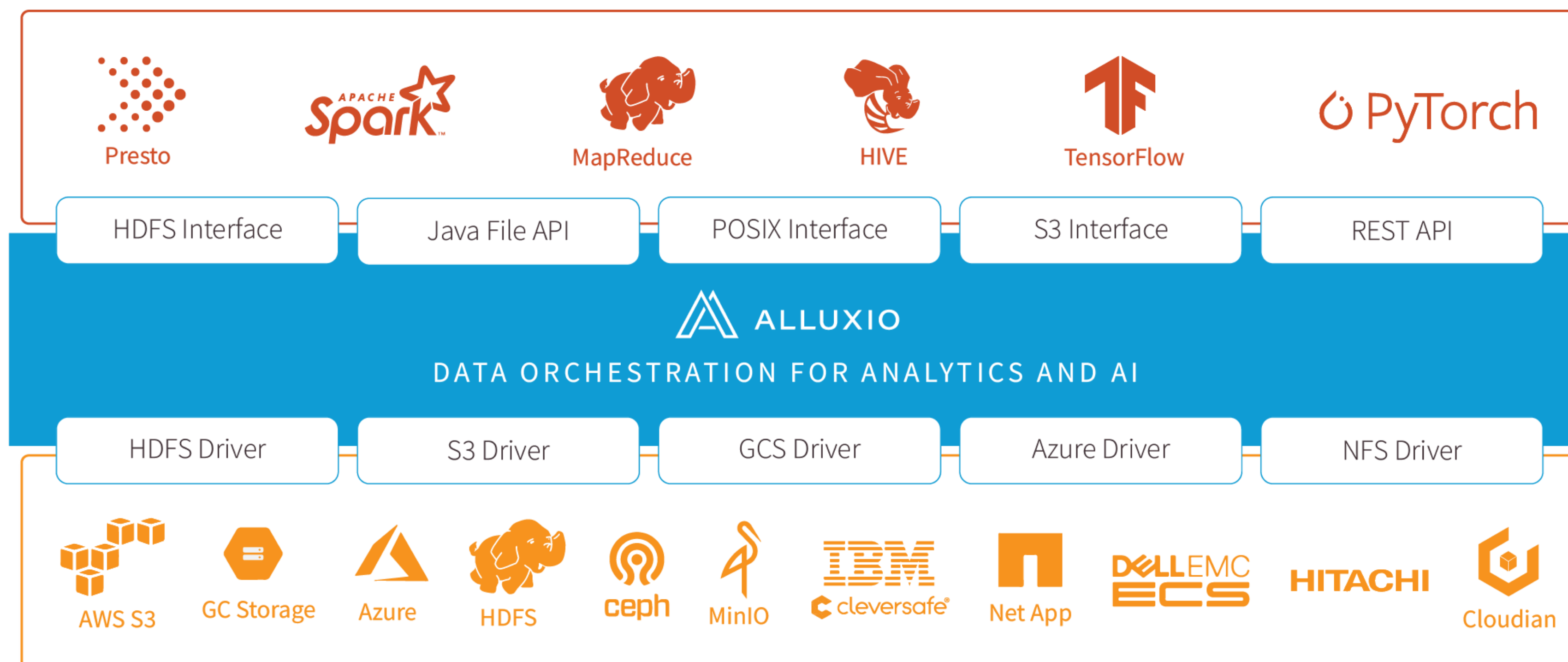


kubernetes

ML 학습 / 인퍼런스

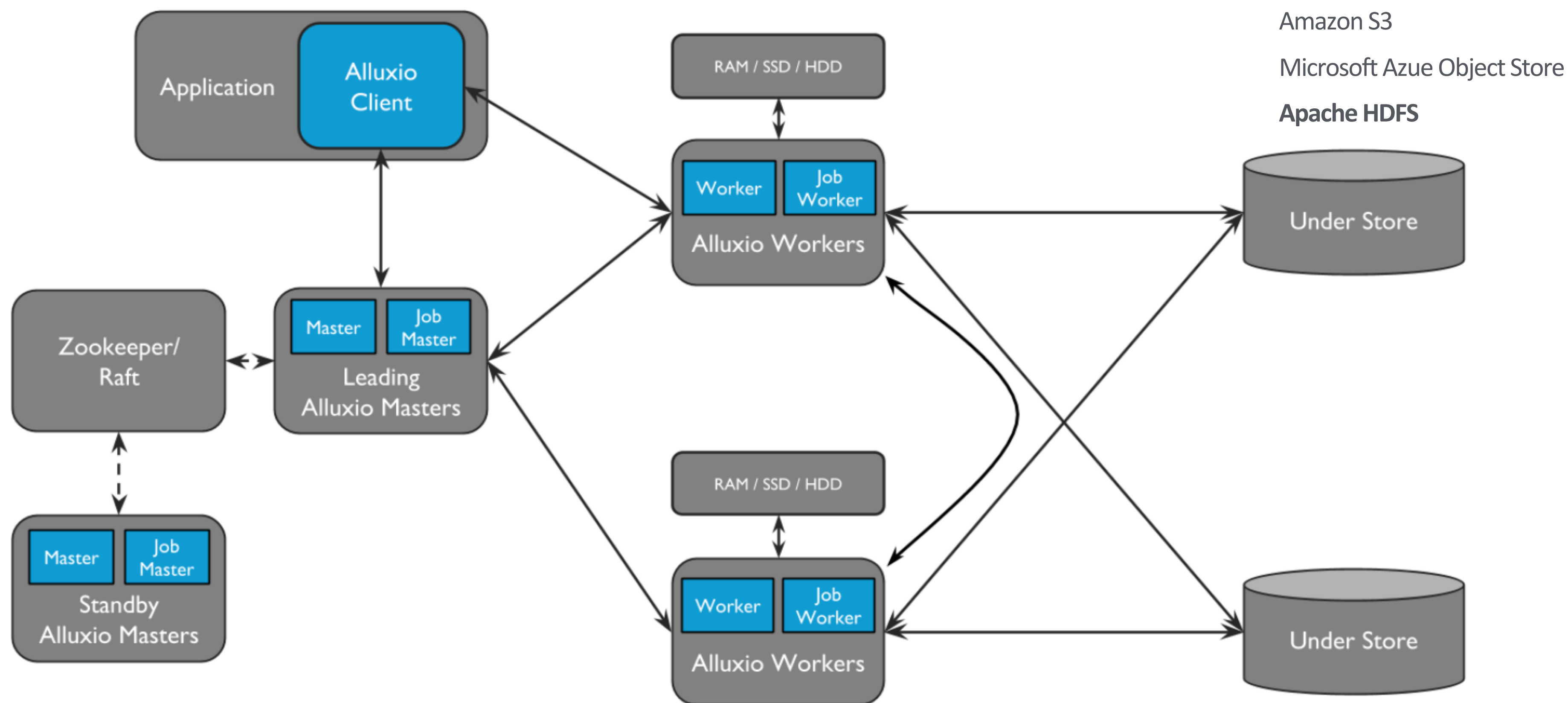
- 따로 storage 클러스터 등을 구축할지
- 기존 사용하던 HDFS를 k8s에서 사용할 수 있을지

5.2 Alluxio



- 물리적으로 멀리 떨어진, 또는 느린 미디어에 저장된 데이터를 캐싱
- 다양한 스토리지를 원하는 인터페이스로 접근

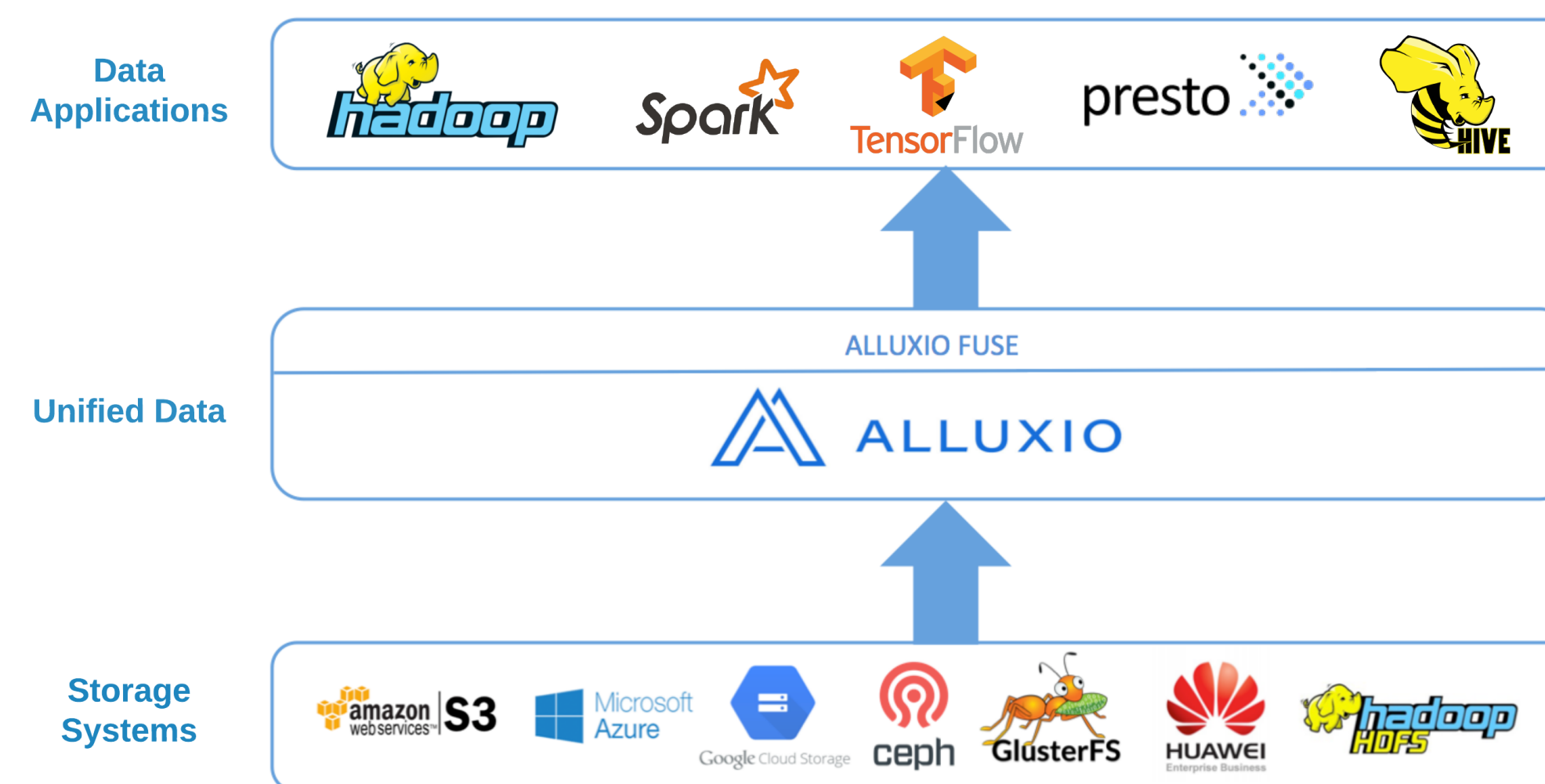
5.2.1 Alluxio의 구조



5.3 Alluxio를 k8s에서 사용하기: Alluxio FUSE?

Alluxio FUSE

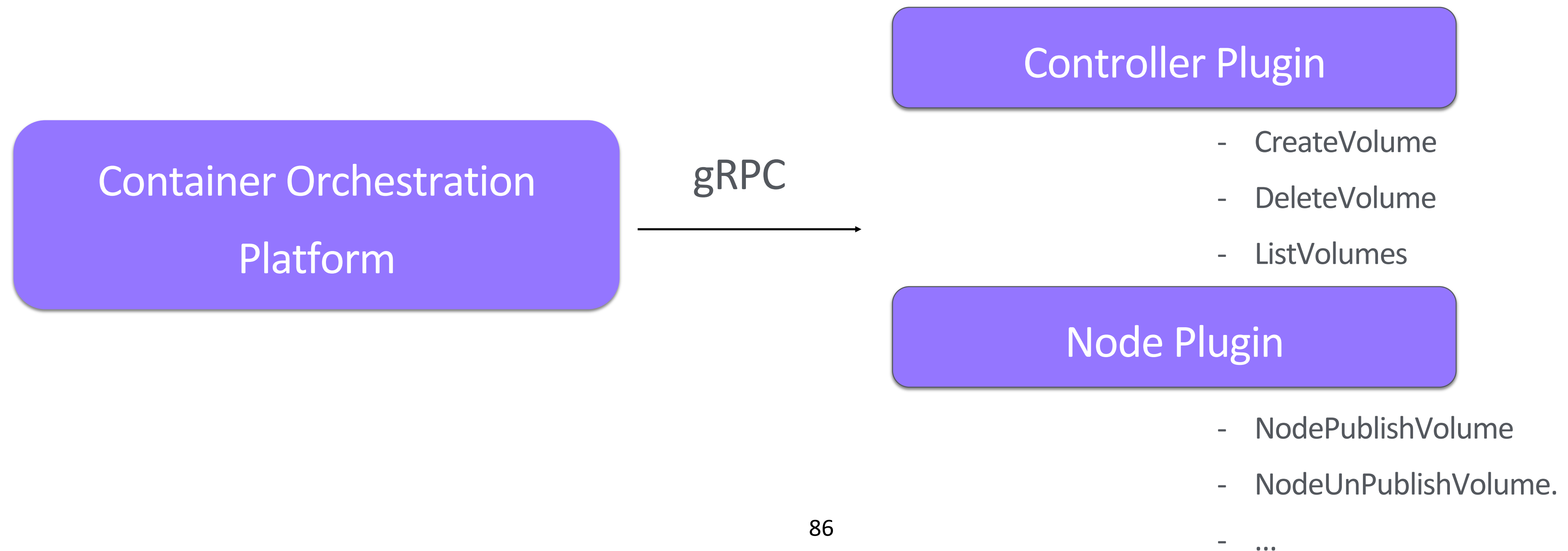
- 각 노드의 특정 경로에 FUSE mount → 사용자는 hostPath volume으로 Alluxio 사용 가능
- 단일 mount point 사용 시 HDFS 권한에 따른 제한 등이 어려워 multi-tenant cluster에 적합하지 않은 방식이라고 판단
- Dynamic provisioning이 어려움



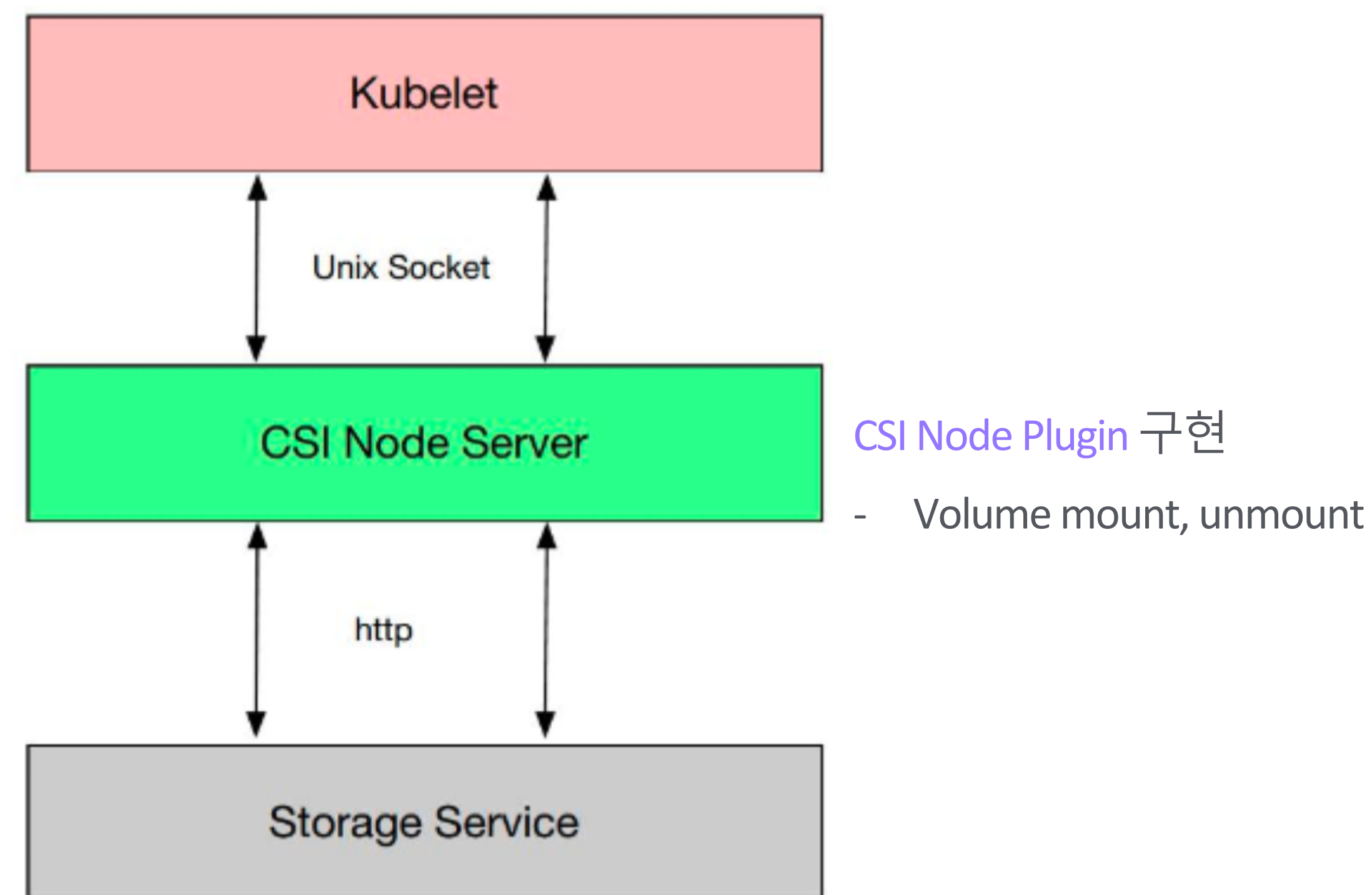
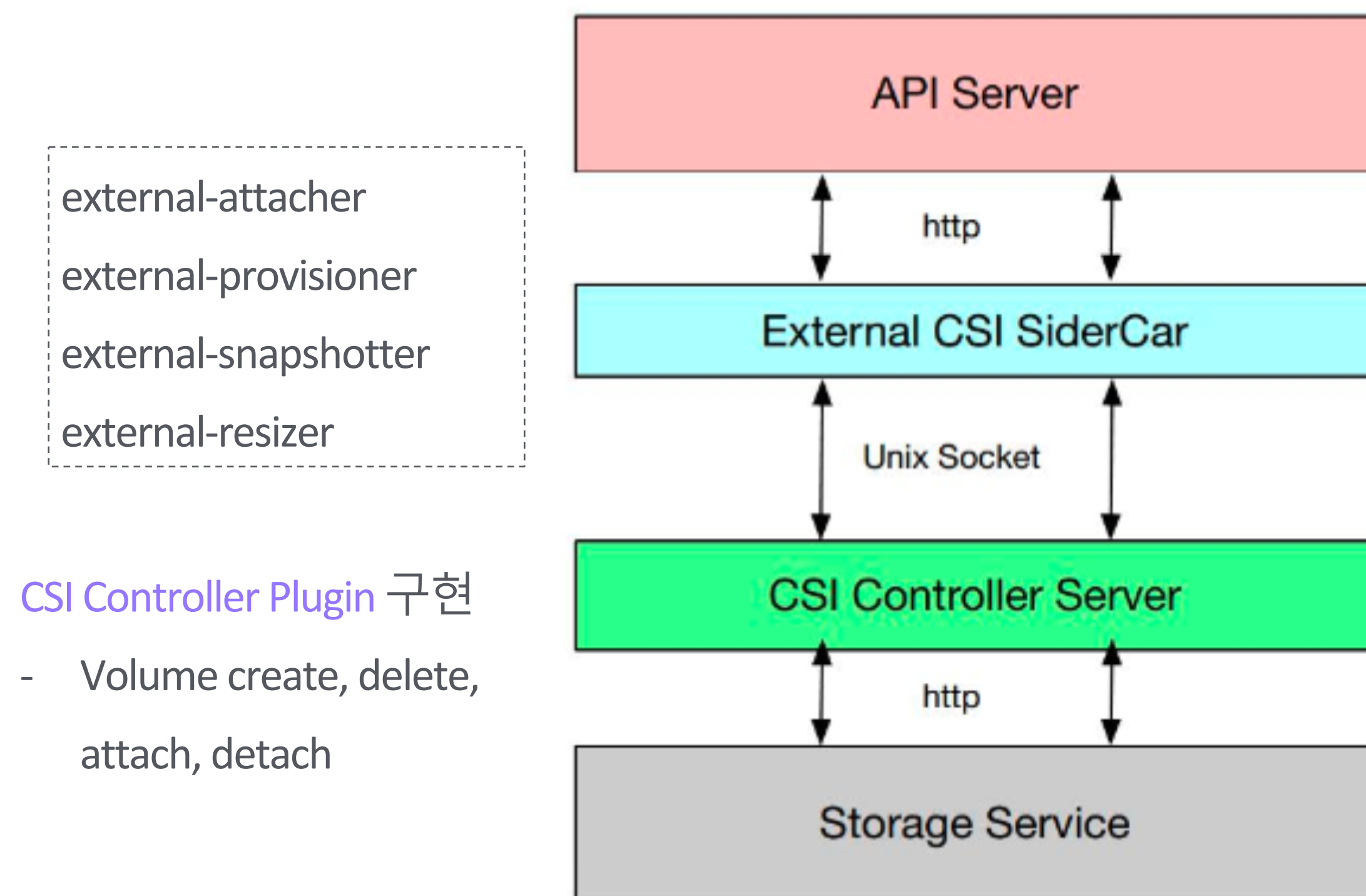
5.4 Alluxio를 k8s에서 사용하기: Alluxio CSI

Container Storage Interface (CSI)

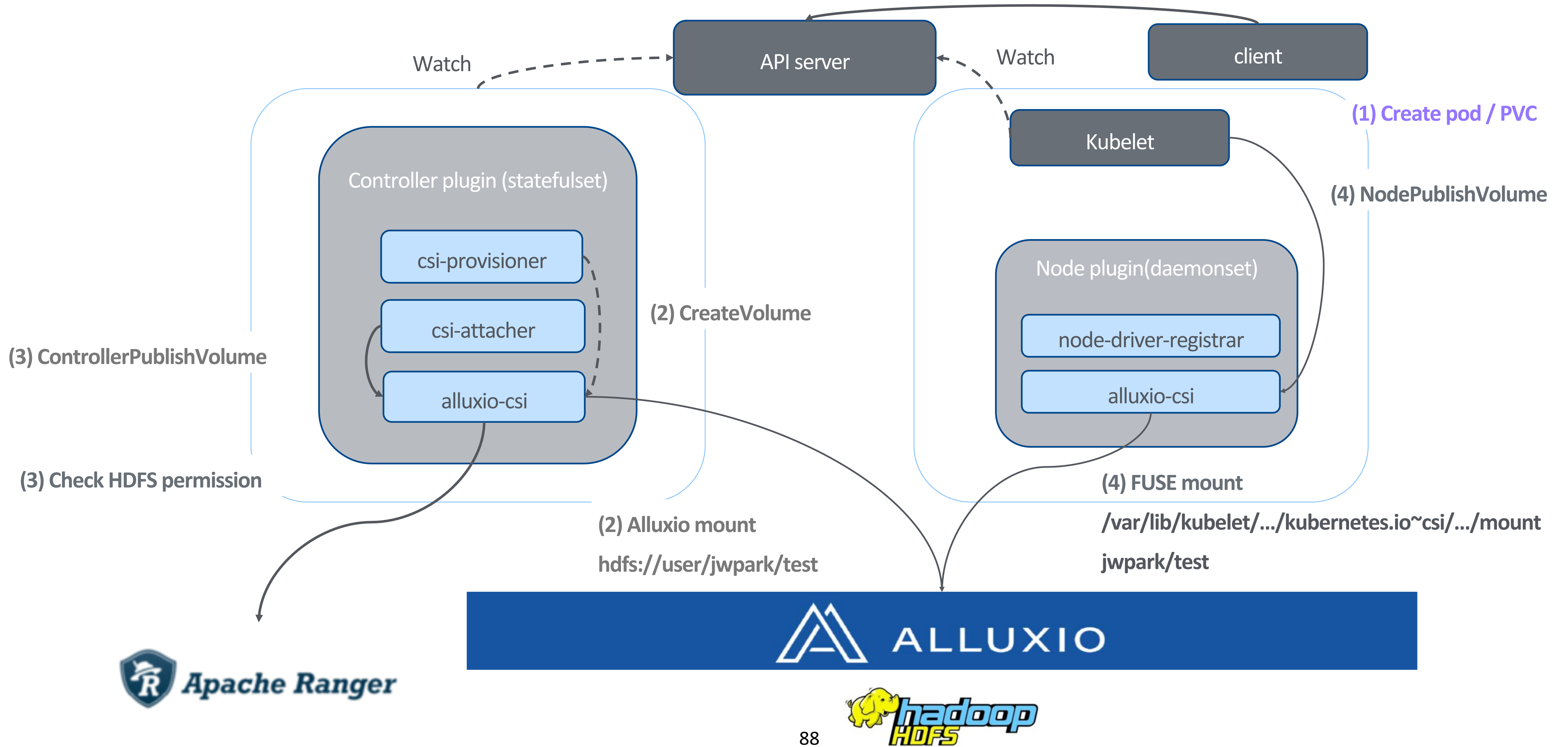
- Storage를 Container Orchestration System에 노출하기 위한 인터페이스 정의
- 어떤 storage든 CSI를 잘 구현하면 k8s volume으로 사용 가능



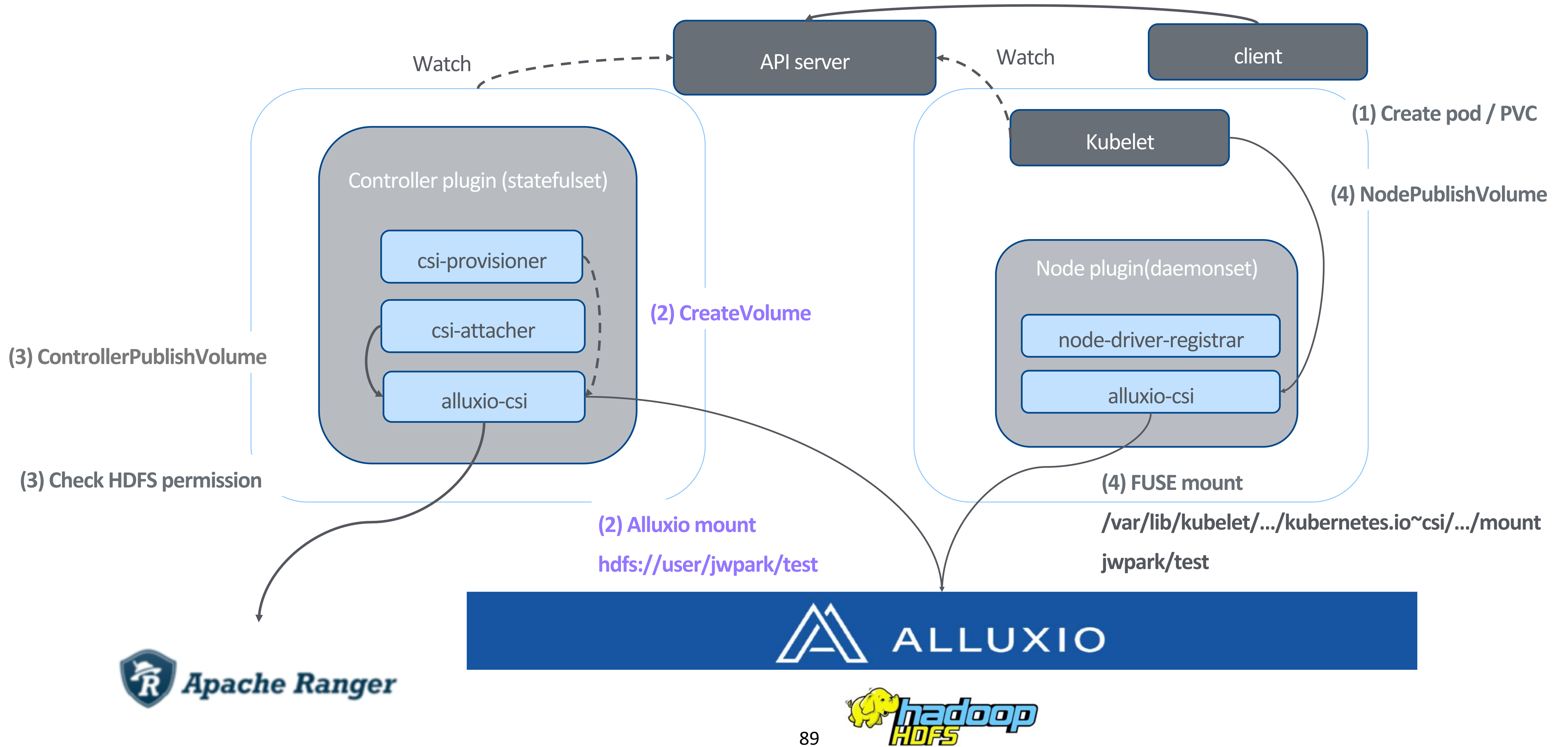
5.4.2 CSI in Kubernetes



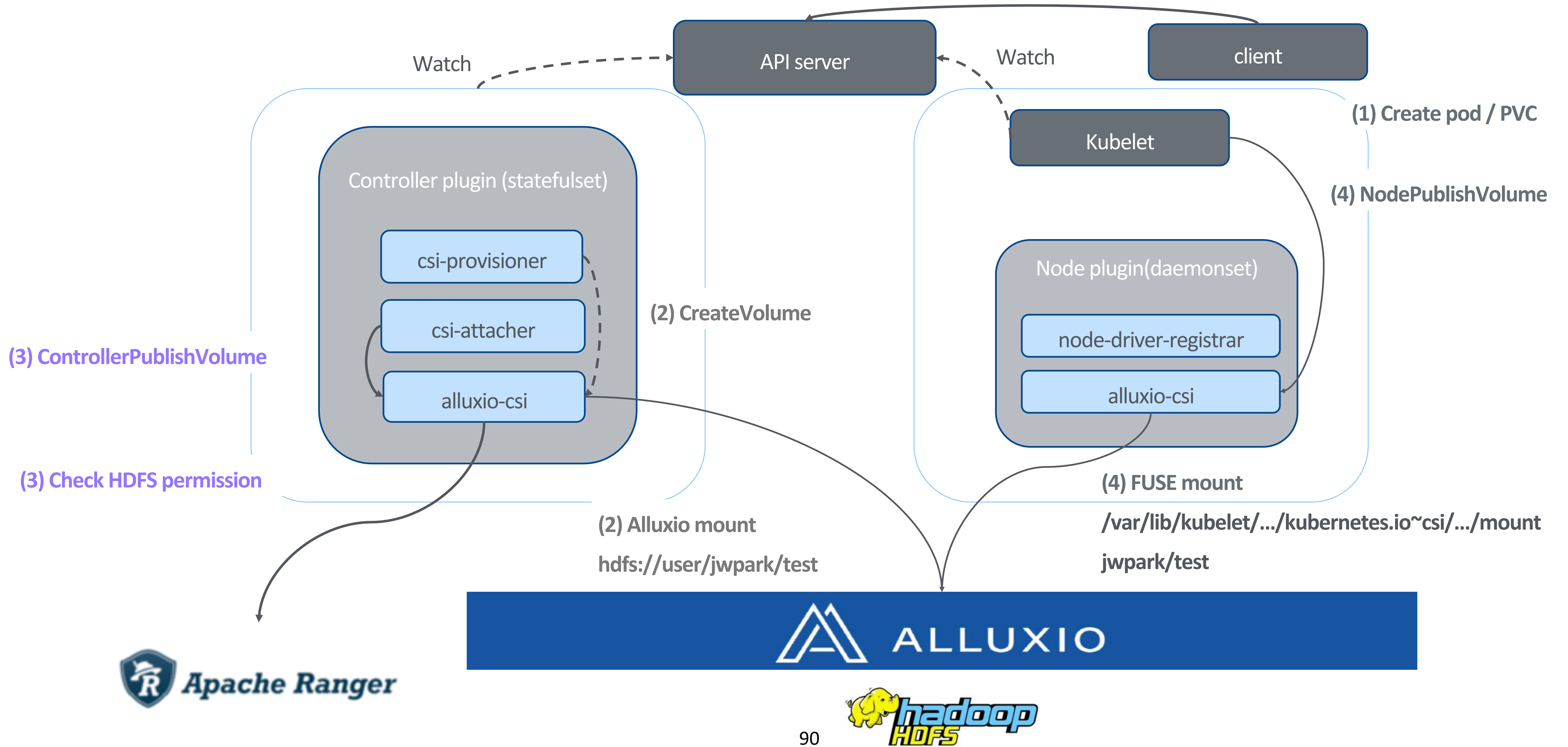
5.4.3 Alluxio CSI: Dynamic Provisioning



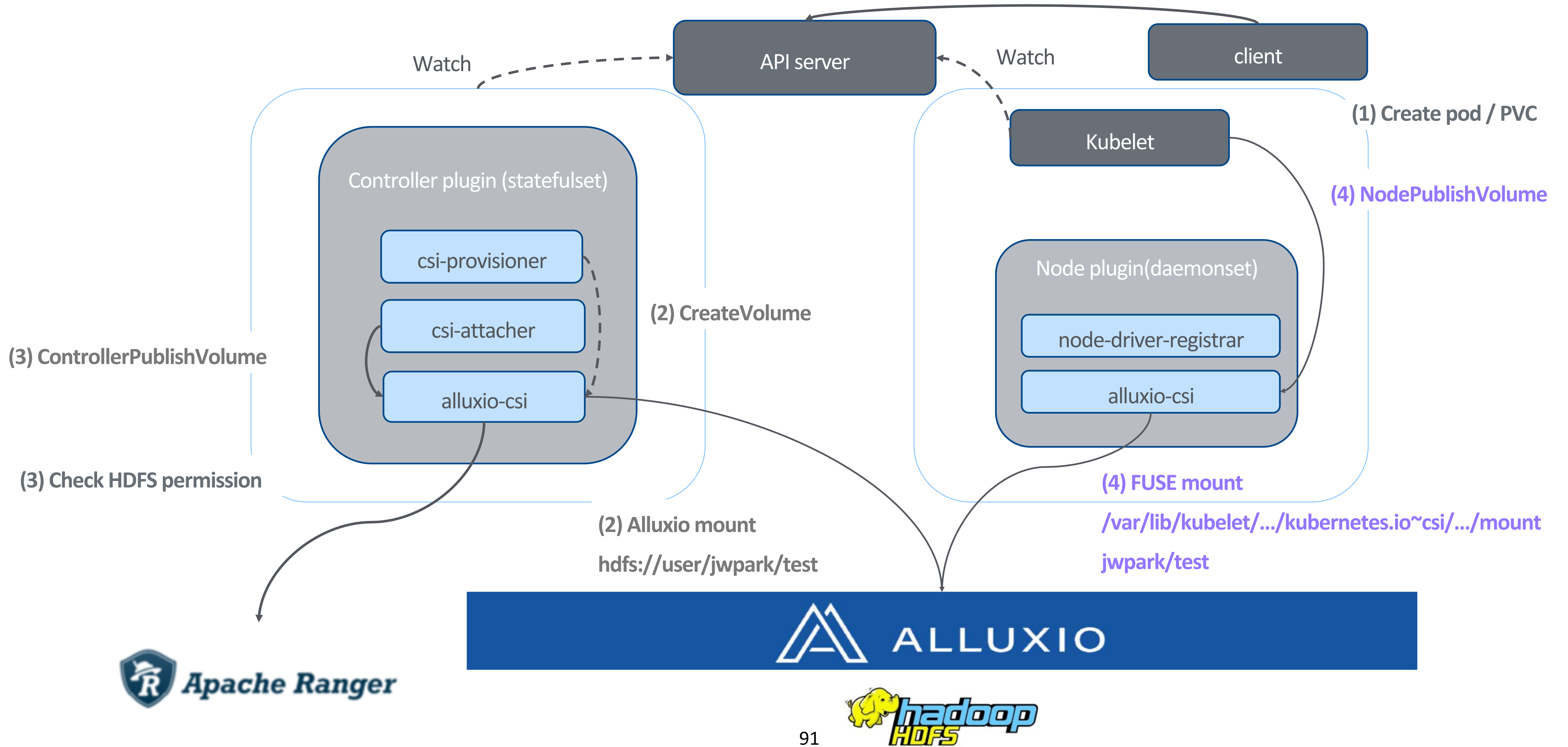
5.4.3 Alluxio CSI: Dynamic Provisioning



5.4.3 Alluxio CSI: Dynamic Provisioning



5.4.3 Alluxio CSI: Dynamic Provisioning



6. Future Work

6.1 더 나은 MLOps를 위해서는?

CI/CD를 개선하고, 다양한 자동화를 적용해야 합니다.

- 파이프라인 개선 (Kale, Elyra 적용 등) 및 소스 관리 자동화
- 이벤트 기반 파이프라인 트리거 기능
- 모델 평가 파이프라인 자동화
- 배포 안정성 테스트 자동화 (인프라 호환성 검증, Locust 및 Cuve Stream을 통한 실트래픽 검증)

또한, 사내 개발자들에게 도움이 되는 유용한 Kubeflow Pipeline 모듈들과 예시들을 제공해 나아갈 예정입니다!

6.2 더 나은 플랫폼을 위해서는?

Multi-Cluster Kubeflow - 클러스터 구축 및 운영 효율화

- 클러스터가 커질수록 운영도 힘들고 control plane에 많은 부하를 줌
- Kubeflow를 여러 Cluster에 빠르고 효율적으로 구축 및 관리하는 방법 필요

GPU Scheduling 개선

- Gang Scheduling
- Topology awareness (inter-node)
- HiveD Scheduler 혹은 volcano 같은 오픈 소스 스케줄링 시스템 도입 고려

6.3 오픈 소스 기여

Kubeflow 관련 project에 아래와 같은 기여들을 해왔어요.

- <https://github.com/kubeflow/manifests/pull/1877>
- <https://github.com/kubeflow/pipelines/pull/5293>
- <https://github.com/kubeflow/pipelines/pull/5552>
- <https://github.com/kubeflow/pipelines/pull/6494>
- <https://github.com/kubeflow/kubeflow/pull/6052>
- <https://github.com/kubeflow/fairing/pull/560>
- <https://github.com/kserve/kserve/pull/1361>
- <https://github.com/kserve/models-web-app/pull/7>
- <https://github.com/kserve/models-web-app/pull/10>
- <https://github.com/NVIDIA/deepops/pull/893>

새로 필요한 기능들을 개발하고, 기존 기능을 개선하여 오픈소스에 기여하는
일을 앞으로도 꾸준히 진행할 예정입니다!

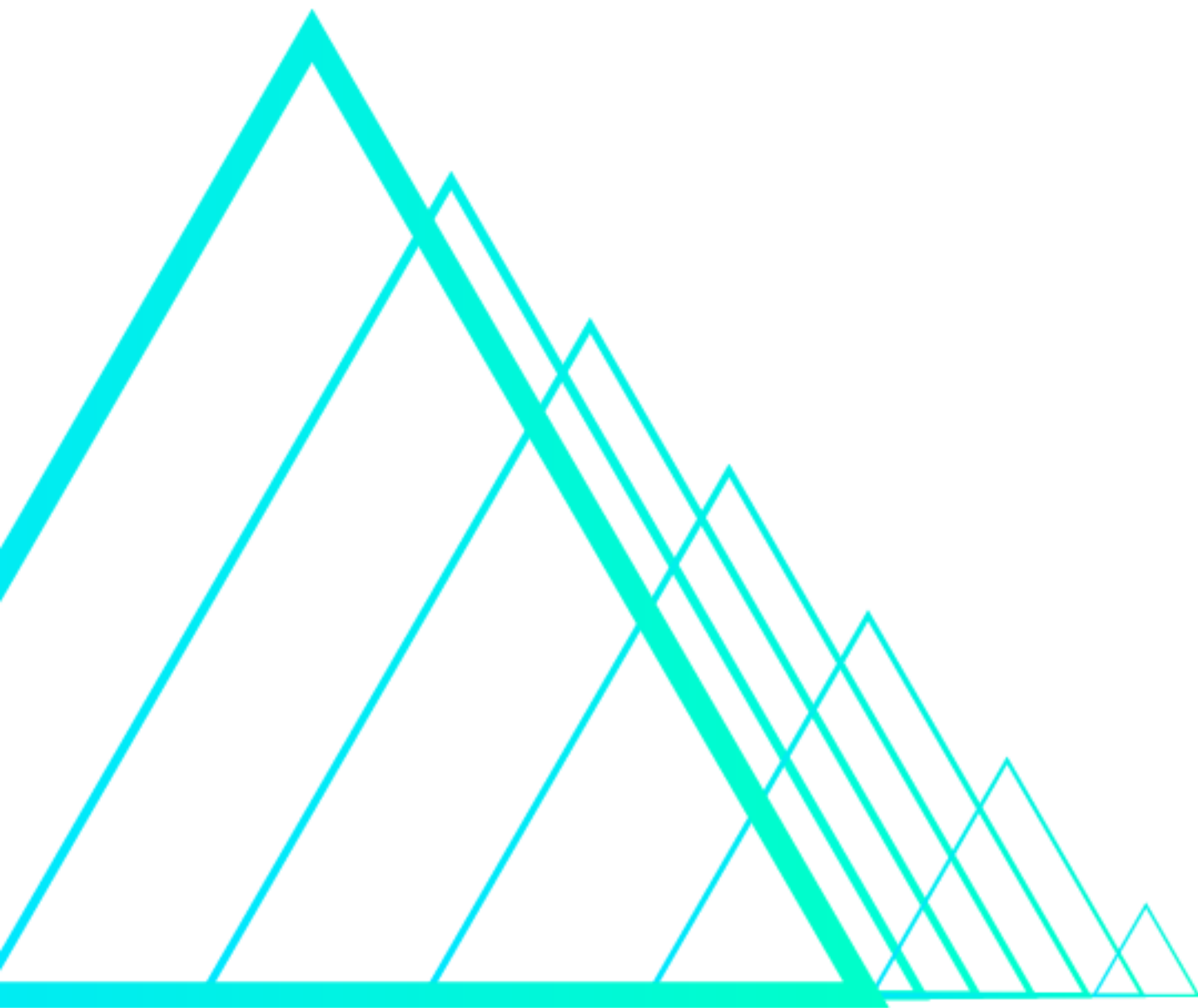
We're hiring

AI & Data Platform 소개

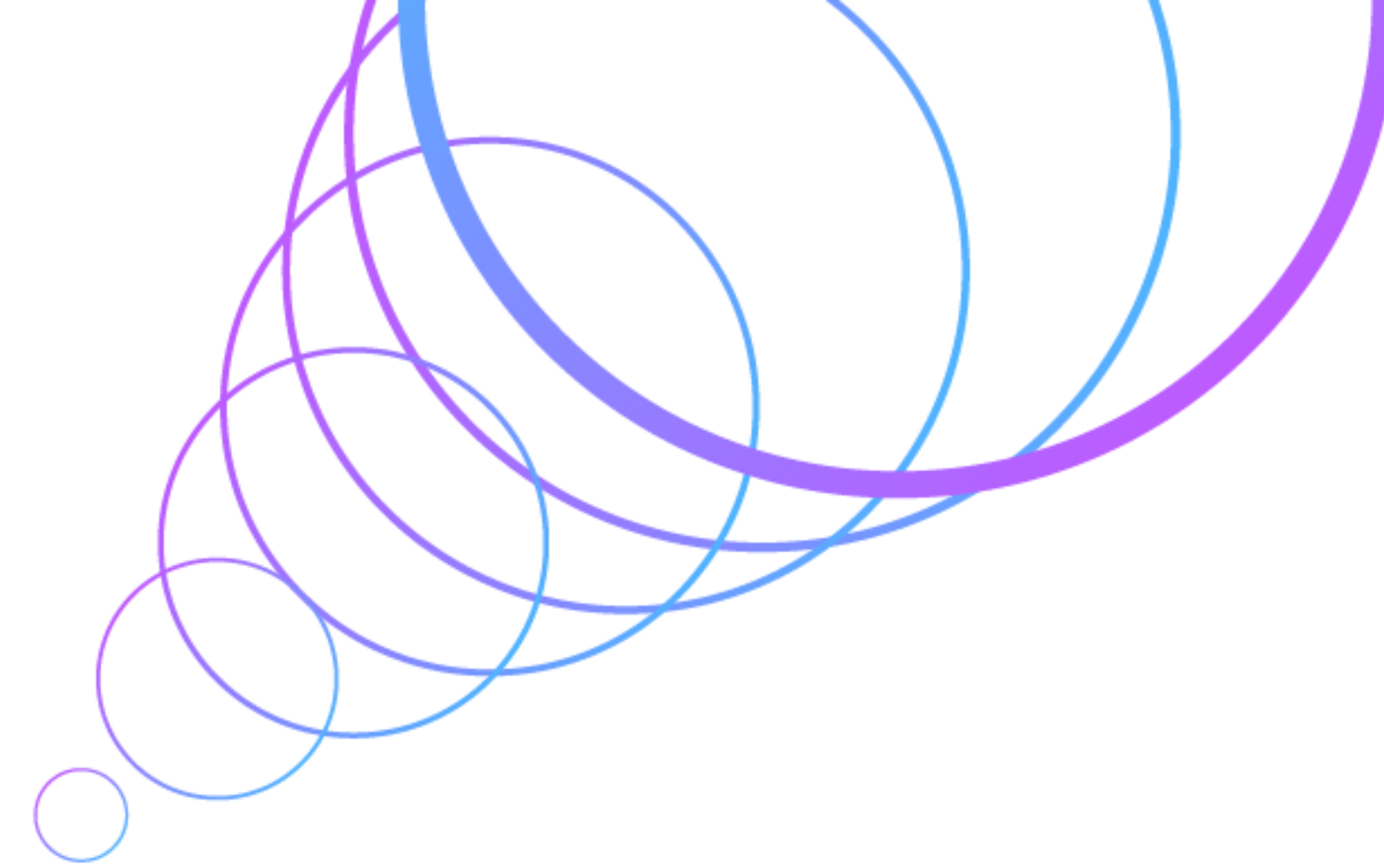
- <https://naver-career.gitbook.io/kr/service/search/ai-and-data-platform>

경력 소개

- <https://d2.naver.com/news/7591059>



Q & A



The slide features several decorative geometric elements. In the top right corner, there is a series of overlapping circles in shades of blue and purple, arranged in a curved, descending pattern. On the left side, there is a series of overlapping triangles in shades of cyan and teal, arranged in a descending, stepped pattern. On the right side, there is a series of overlapping parallelograms in shades of orange, red, and pink, arranged in a descending, stepped pattern. The text "Thank You" is centered in the middle of the slide in a large, black, sans-serif font.

Thank You