우리FISA 클라우드 엔지니어링 3기 기술 세미나

AI 모델 배포를 위한 KSERVE 모델 서빙 환경 구축하기

허예은

우리에프아이에스아카데미

목차

CONTENT

PART 1

주제 선정 배경

주제 선정 배경

- Kubeflow와 KServe에 대한 이해
- 타 프레임워크와의 비교

PART 2

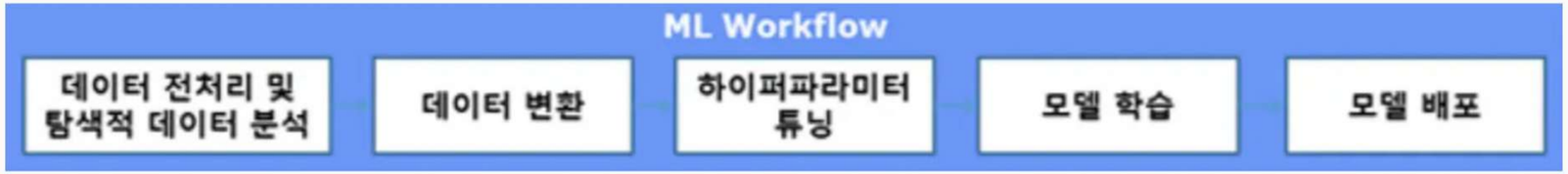
KUBEFLOW

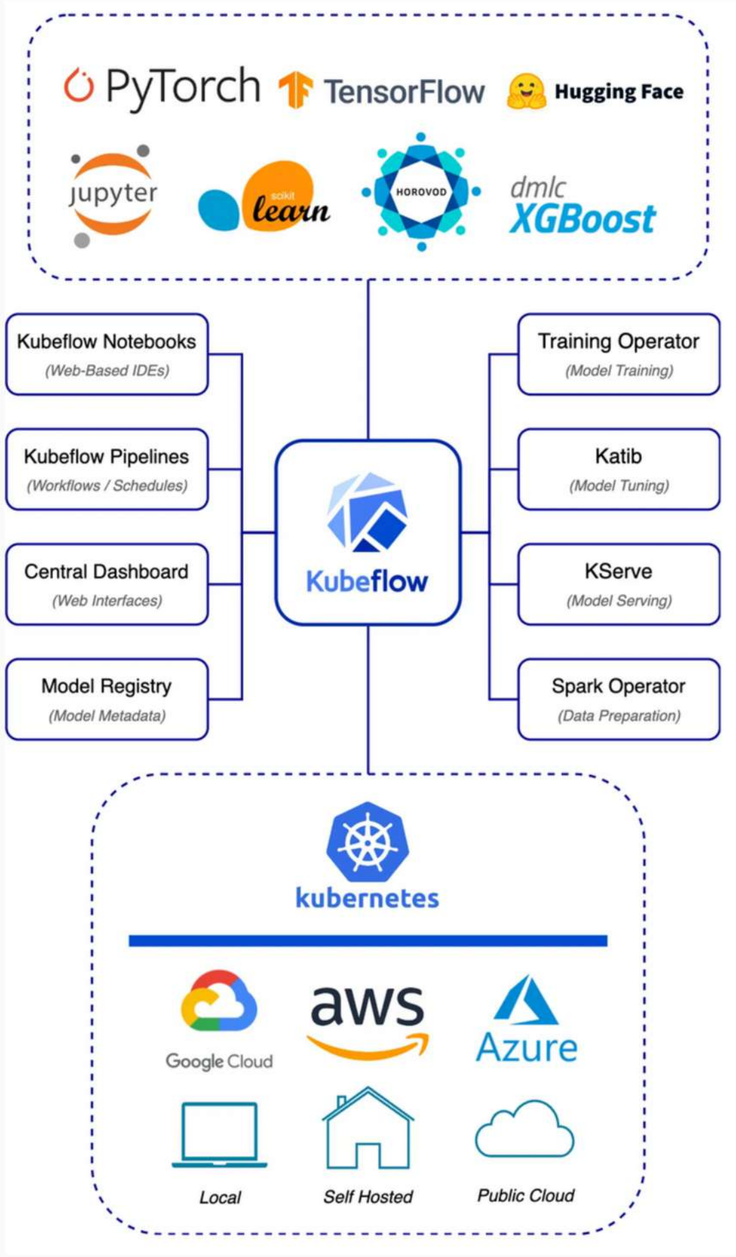
Kubeflow란?

- 엔드투엔드(End-to-End) AI 플랫폼
- 머신러닝 모델 학습부터 배포 단계까지 머신러닝 워크플로우의 모든 작업에 필요한 도구와 환경을 Kubernetes 위에서 컴포넌트로 제공

AI 플랫폼

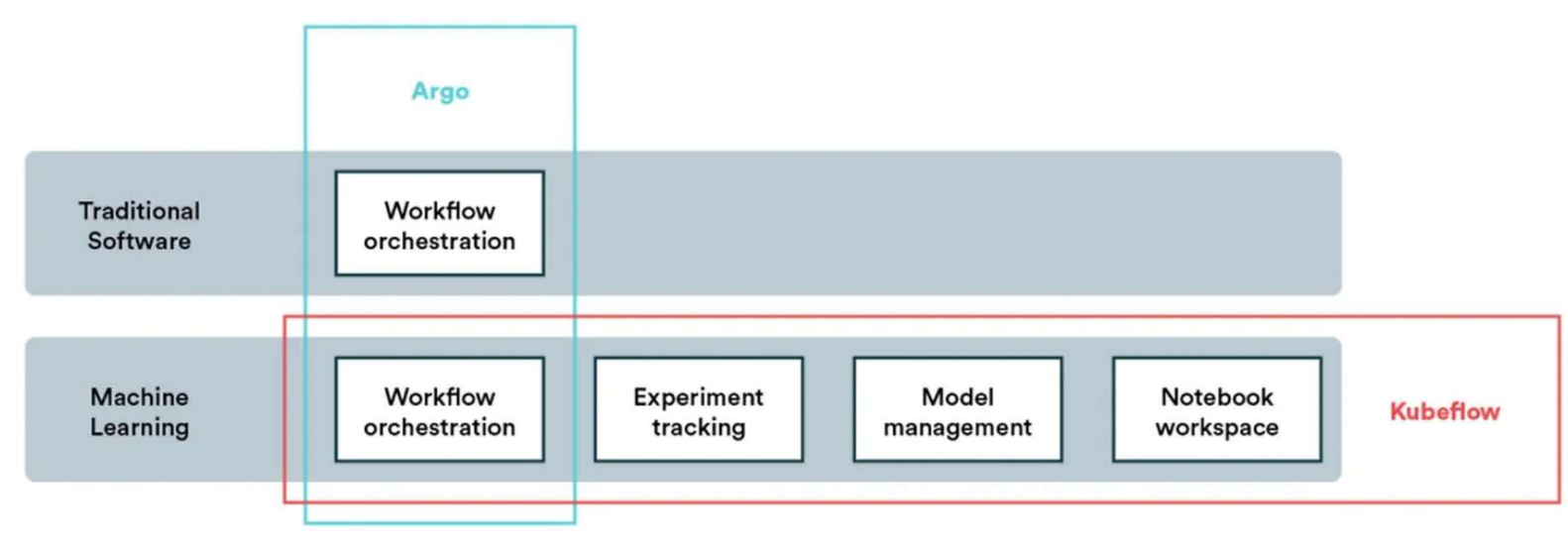
- 머신러닝 워크플로우를 자동화
- AI 플랫폼은 각 단계에서 모델 개발에 필요한 도구 및 환경을 수작업으로 구축하는 작업을 최소화
- 퍼블릭 클라우드 - 구글 버텍스 AI(Google Vertex AI), 아마존 세이지메이커(Amazon SageMaker), 애저 머신러닝(Azure Machine Learning)
- 오픈소스 - 쿠브플로우(Kubeflow)





타 프레임워크와 비교 - Argo Workflow

- Argo Workflow : Workflow Orchestration Only
- Kubeflow : Workflow Orchestration 뿐만 아니라 머신러닝 워크플로우 기능 제공



타 프레임워크와 비교 - Amazon SageMaker

Kurly만의 MLOps 구축하기 - 쿠브플로우 도입기

쿠브플로우를 도입한 이유와 유용한 팁

	세이지 메이커(SageMaker)	쿠브플로우 (Kubeflow)
데이터 라벨링	Ground Truth	-
전처리	Data Wrangler	-
실험 및 학습	Studio Notebooks & Notebook Instances	Kubeflow Noteoobk
하이퍼파라미터 튜닝	Automatic Model Tuning	Katib
워크플로우	SageMaker Pipelines	Kubeflow Pipelines
학습 모니터링	Model Monitor	Tensorboard
서빙	Endpoints	KServe, BentoML
피쳐스토어	Feature Store	Feast

- SageMaker
  - Fully-managed 서비스로 Kubeflow에 비해 머신러닝 라이프 사이클의 더 큰 영역을 커버
  - 그러나 비용 문제와 함께 기술 내재화 측면에서 좋지 않음
- Kubenetes를 적극 활용하고 있는 상황이었던 컬리 데이터 플랫폼팀은 Kubeflow 선택!
  - Kubeflow가 커버하지 못하는 머신러닝 라이프 사이클 영역은 직접 개발하거나 다른 오픈 소스 툴을 활용하여 충분히 극복 가능할 것이라고 판단

PART 3

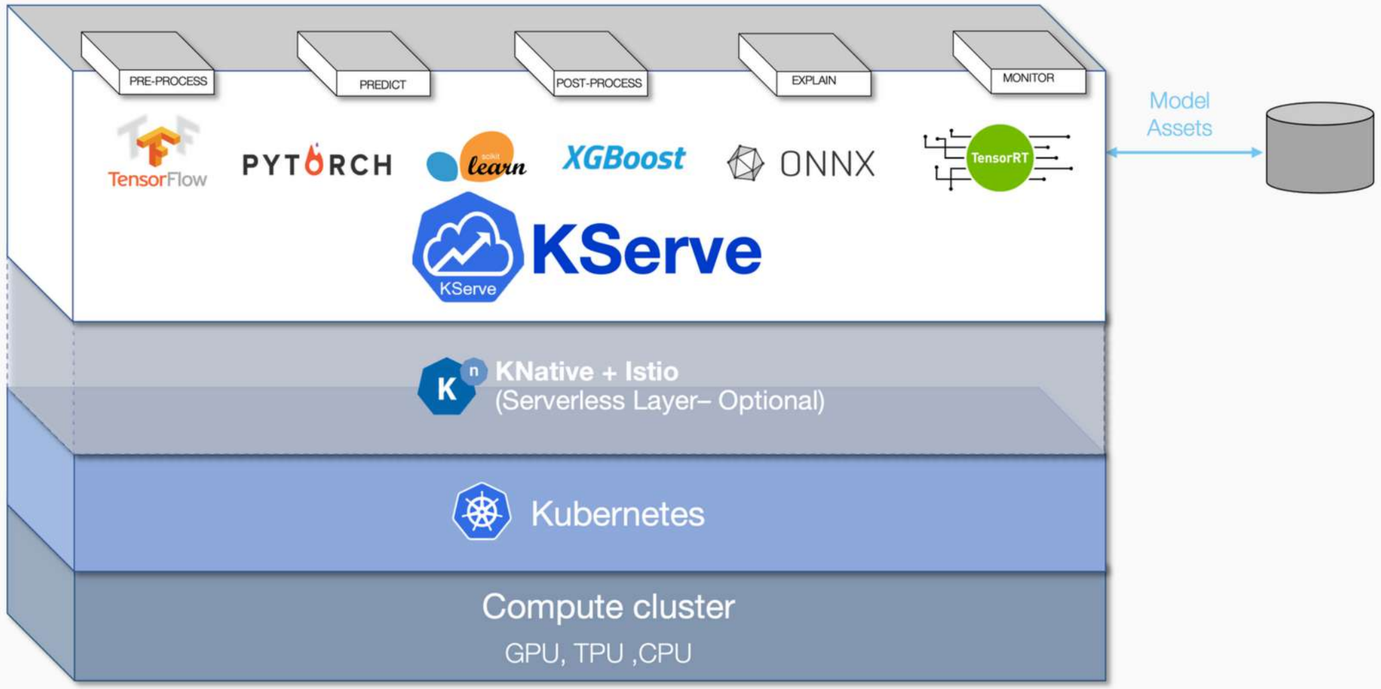
KSERVE

KServe란?

- 다중 프레임워크를 지원하는 확장성 높은 모델 추론 플랫폼
  - 추론 서버를 InferenceService라는 Custom Resource로 추상화해서 관리
  - Runtime agnostic한 관리를 제공

" MLOps를 위해서는 Language/Framework Agnostic한
Model Serving을 위한 General Framework이 필요하고,
현 시대의 Defacto는 Kubernetes며,
현자들이 이미 Kubernetes 기반으로 필요한 소프트웨어를 Package화 했으니
이것이 KServe다. "

KServe란?

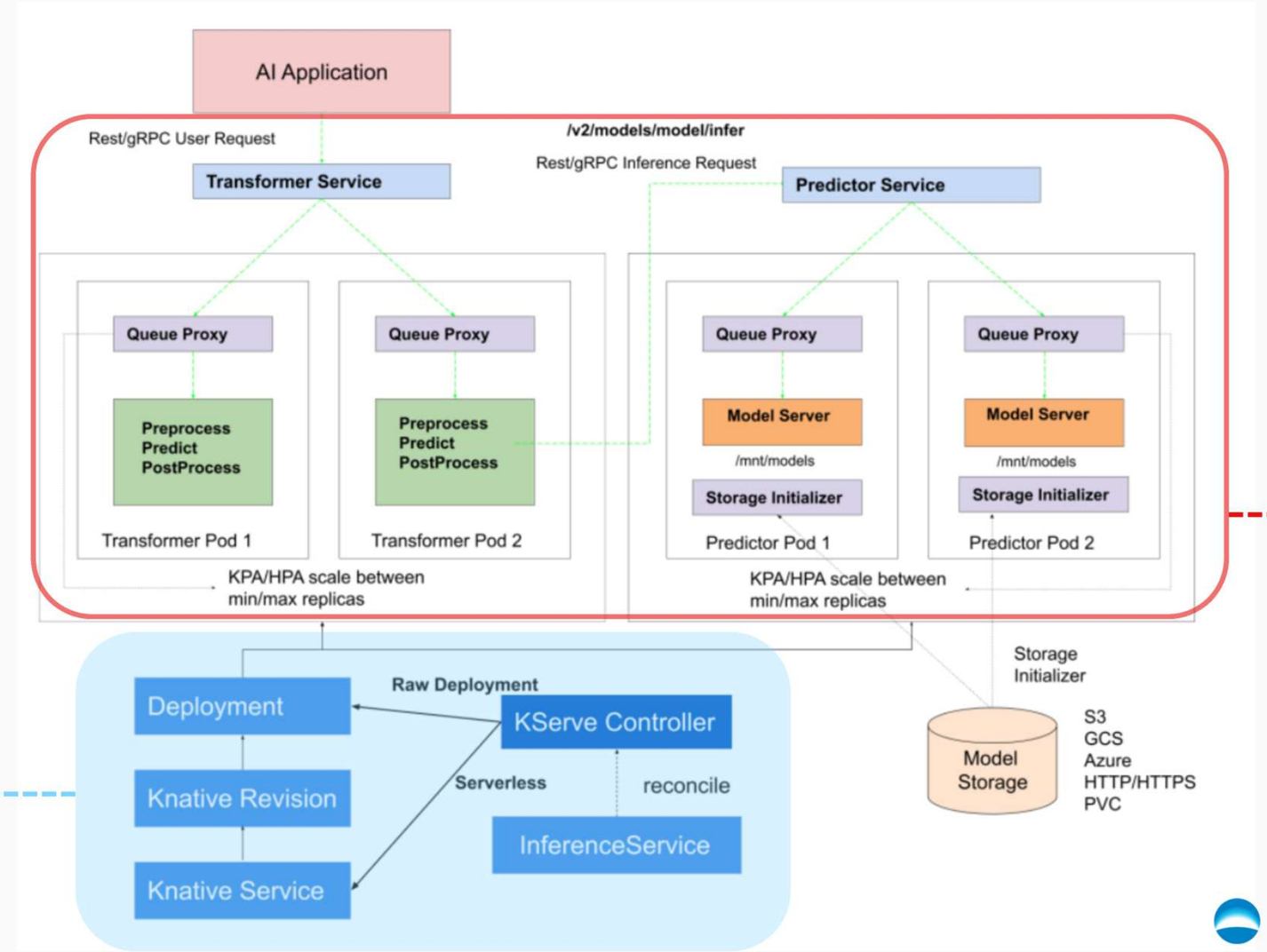


- Knative
  - 자동 스케일업 및 스케일다운 기능을 갖춘 서버리스 모드를 지원
  - 서버리스: 요청이 있을 때만 자원을 사용하고, 요청이 없을 때는 자원을 해제해 비용을 절감
- Istio
  - 서비스 엔드포인트를 API 사용자에게 공개하기 위한 인그레스로 사용

KServe 기능

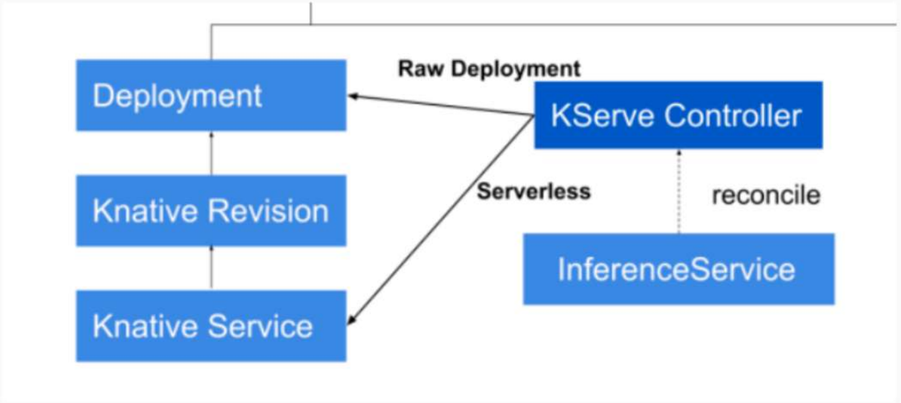
- 대규모 확장성이 필요한 사용 사례에 맞춰 설계됨
- 머신러닝 프레임워크 전반에서 성능이 우수하고 표준화된 추론 프로토콜을 제공
- GPU를 포함한 서버리스 추론 작업을 지원
- Autoscaling과 트래픽이 없을 때는 Scale-to-Zero 기능을 제공
- 예측, 사전/사후 처리, 모니터링을 포함한 간단하고 플러그 가능한 프로덕션 서빙 환경을 제공
- Canary 배포, 앙상블(ensemble), 트랜스포머(transformer)를 포함한 고급 배포 기능을 제공

KServe 아키텍처


AI Application
Rest/gRPC User Request
/v2/models/model/infer
Rest/gRPC Inference Request
Transformer Service
Predictor Service
Queue Proxy
Queue Proxy
Queue Proxy
Queue Proxy
Preprocess
Predict
PostProcess
Preprocess
Predict
PostProcess
Model Server
/mnt/models
Storage Initializer
Model Server
/mnt/models
Storage Initializer
Transformer Pod 1
Transformer Pod 2
Predictor Pod 1
Predictor Pod 2
KPA/HPA scale between min/max replicas
KPA/HPA scale between min/max replicas
Data Plane
Storage Initializer
Raw Deployment
Deployment
KServe Controller
Knative Revision
Serverless
reconcile
InferenceService
Knative Service
Model Storage
S3
GCS
Azure
HTTP/HTTPS
PVC
Control Plane

KServe 아키텍처

Control Plane

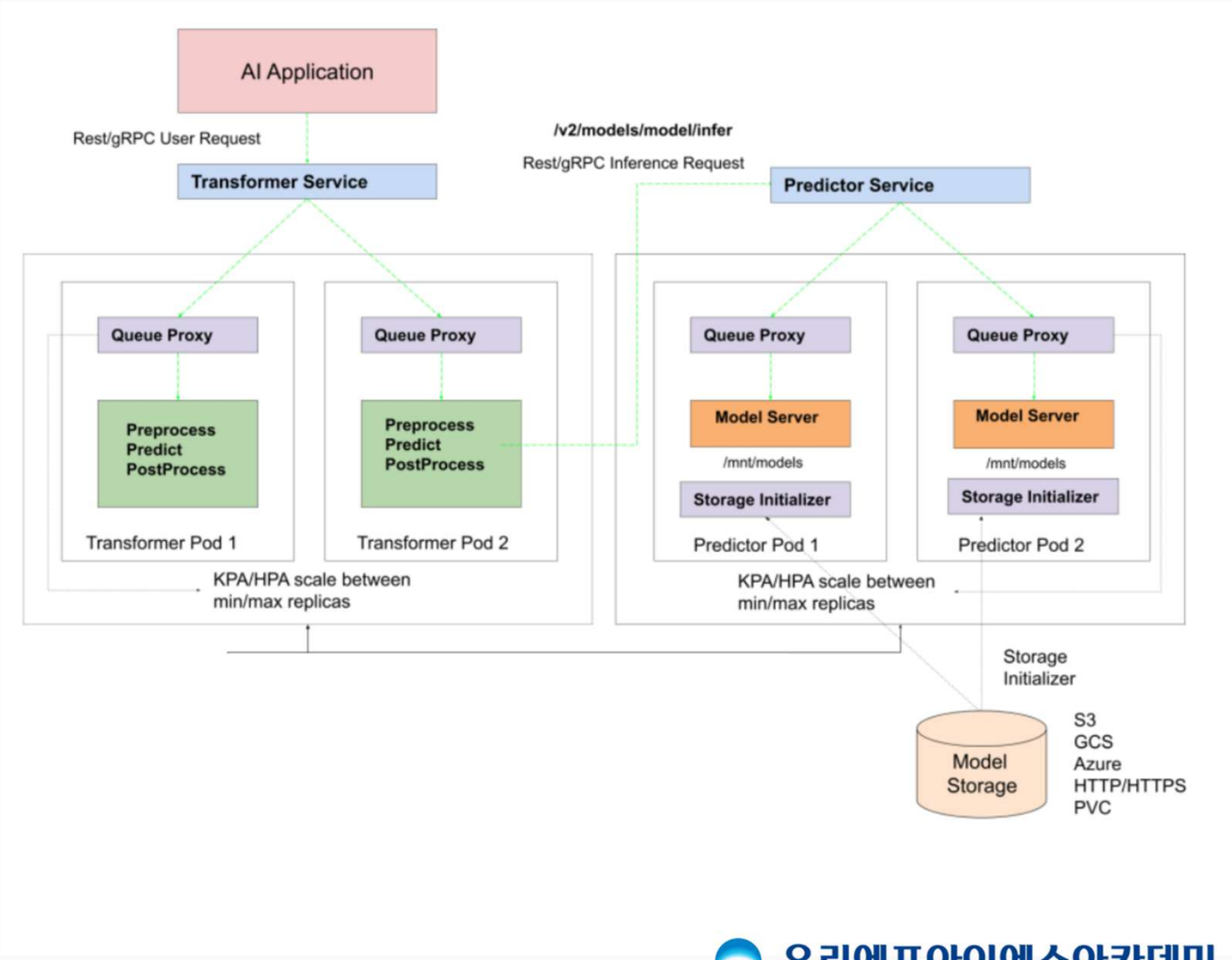


- 역할
  - KServe Controller로 InferenceService의 라이프 사이클을 관리
    - InferenceService : 추론을 담당하는 커스텀 리소스
- Knative와의 연계
  - 대표적으로 서버리스 모드를 지원해, Knative 리소스와 연계하여 모델을 배포
  - GPU 기반 워크로드의 효율적 처리가 가능
  - Canary 배포와 같은 고급 배포 전략을 지원
  - 요청이 없을 때 Pod를 0으로 줄이는 기능인 Scale-to-Zero를 지원
  - Raw Deployment 모드에서 HPA는 사용할 수 있지만 Scale-to-Zero는 지원하지 않음

KServe 아키텍처

Data Plane

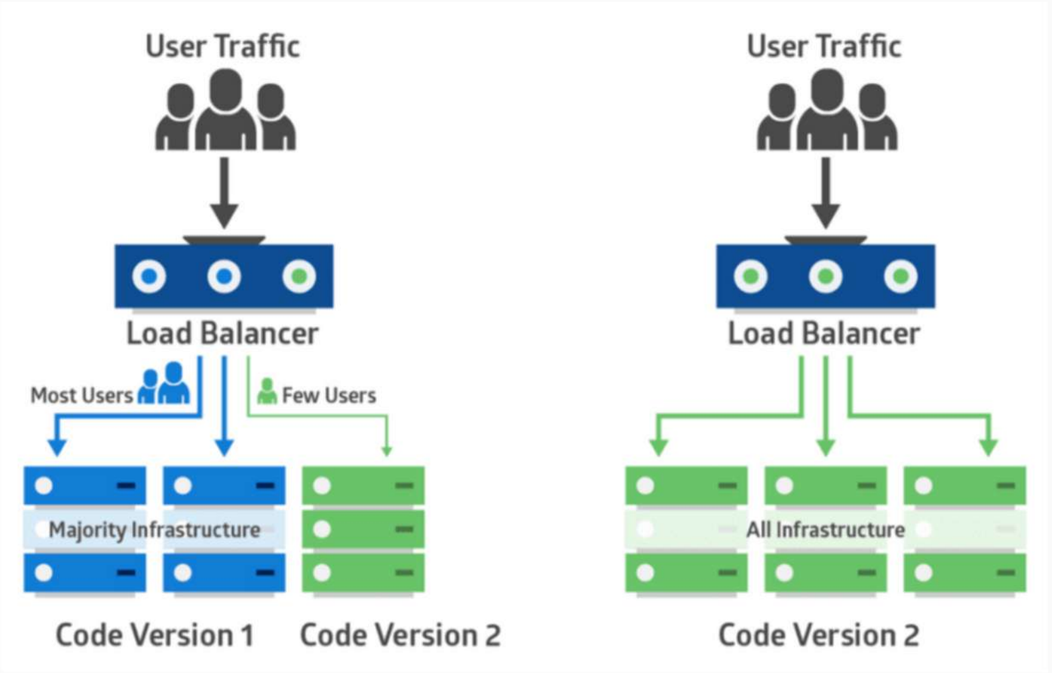
- 역할
  - 실질적으로 모델의 서빙을 담당하는 구성 요소
  - 사용자의 요청을 처리하고 예측 결과를 반환
- Transformer Service
  - 요청 받은 입력 데이터를 전처리 및 후처리
- Predictor Service
  - 실제 머신러닝 모델이 실행되는 부분
- Model Server & Storage Initializer
  - 모델이 저장되어 있는 곳에서 추론을 수행
  - 모델 서빙을 담당하는 핵심 컴포넌트
  - 모델 스토리지(S3, GSC, PVC)로부터 모델 파일을 로드



KServe 기능 - Canary 배포

Canary 배포

- 전략
  - 새로운 버전을 작은 비율로 배포해 성능 모니터링을 한 후, 문제가 없을 경우 점차 트래픽 비율을 늘리는 방식
  - 위험을 최소화하면서 새로운 모델을 배포하는 전략
- 효과
  - 성능이 안정적임을 확인한 후 전체적으로 확장할 수 있음
- 트래픽 할당
  - 모델 업데이트 시 트래픽의 일부만 새로운 모델로 보내고, 성능이 안정적이면 점진적으로 더 많은 트래픽을 할당하게 됨
- KServe에서의 활용
  - KServe는 모델 버전 관리를 위해 Canary 배포 전략을 지원함



PART 4

AI 모델 배포를 위한 서빙 환경 구축

Environments

- 클러스터 구성 : 마스터 노드 (10.0.2.21), 워커 노드 2개 (10.0.2.22, 10.0.2.23)
- OS : Ubuntu 22.04
- Kubernetes : v1.31
- KServe : v0.13

1. KServe 설치

0) 클러스터 구축

- 각 노드에서 kubeadm, kubelet, kubectl을 v1.31으로 통일하여 설치
- 마스터 노드, 워커 노드 설정

1) KServe 설치

- 최신 버전인 v0.13 설치

```
# KServe Quick installation script on Kind
curl -s "https://raw.githubusercontent.com/kserve/kserve/release-0.13/hack/quick_install.sh" | bash
```

1. KServe 설치

2) 실행 중인 Pod 확인

- kserve-controller-manager
  - KServe의 중앙 제어 컴포넌트
  - 모델 배포와 관련된 주요 로직을 담당

- istio-ingressgateway
  - 클러스터 외부에서 들어오는 트래픽을 수신하는 역할
  - KServe에서 모델 서빙 요청 시, 외부 트래픽은 Ingress Gateway를 통해 들어와 Istio 내부의 서비스로 전달

- autoscaler
  - 트래픽의 양을 모니터링해 서비스의 Pod 수를 자동으로 조정
  - 트래픽이 많으면 Pod 수를 증가시키고, 트래픽이 적을 때는 Pod 수를 줄여 리소스를 절약

```
username@myserver02:~$ kubectl get pods -A
NAMESPACE         NAME                                         READY   STATUS
cert-manager      cert-manager-7b55dcfbb4-r7526                1/1     Running
cert-manager      cert-manager-cainjector-5b4555d545-xknx4     1/1     Running
cert-manager      cert-manager-webhook-679949d4cb-h6xcf        1/1     Running
istio-system      istio-ingressgateway-5f8b7bf488-qbwt6        1/1     Running
istio-system      istiod-88d5cf95b-nnhcc                       1/1     Running
knative-serving   activator-b8cfc79c9-52rrp                    1/1     Running
knative-serving   autoscaler-5fd4ccfc-22jb7                    1/1     Running
knative-serving   controller-756fdbd98c-5pvrv                  1/1     Running
knative-serving   net-istio-controller-8494d6ddbd-zhqqb        1/1     Running
knative-serving   net-istio-webhook-77f6678db-8lwcd            1/1     Running
knative-serving   webhook-7c5c565449-v57rb                     1/1     Running
kserve            kserve-controller-manager-76fb78ffb8-bvsgv   2/2     Running
kube-system       calico-kube-controllers-6879d4fcdc-xvbmw     1/1     Running
kube-system       calico-node-drgjp                            1/1     Running
kube-system       calico-node-phzzs                            1/1     Running
kube-system       calico-node-szrgs                            1/1     Running
kube-system       coredns-7c65d6cfc9-mxlpq                     1/1     Running
kube-system       coredns-7c65d6cfc9-sqk72                     1/1     Running
kube-system       etcd-myserver01                              1/1     Running
kube-system       kube-apiserver-myserver01                    1/1     Running
kube-system       kube-controller-manager-myserver01           1/1     Running
kube-system       kube-proxy-4w89q                             1/1     Running
kube-system       kube-proxy-btb2x                             1/1     Running
kube-system       kube-proxy-jg9bt                             1/1     Running
kube-system       kube-scheduler-myserver01                    1/1     Running
```

2. InferenceService 파일 작성

InferenceService

- Kubernetes에서 제공하는 커스텀 리소스(CRD)
- 모델 서빙의 모든 단계를 관리하는 KServe의 핵심 리소스

```yaml
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "tf-flower"
  namespace: "kserve-test"
spec:
  predictor:
    model:
      modelFormat:
        name: "tensorflow"
      storageUri: "gs://kfserving-examples/models/tensorflow/flowers"
      resources:
        requests:
          cpu: "500m"
          memory: "1Gi"
```

- modelFormat
  - 서빙할 모델의 형식을 지정
  - TensorFlow, PyTorch, ONNX, Scikit-learn, XGBoost 등의 여러 머신러닝 프레임워크를 지원
- storageUri
  - 모델이 저장된 경로 지정
  - 주로 클라우드 스토리지(GCS, S3 등)를 사용
  - 로컬에서 모델을 제공하는 경우에는 PVC(Persistent Volume Claim) 설정 필요

2. InferenceService 파일 작성

tensorflow/flowers

- 꽃 이미지를 분류하는 딥러닝 모델
- TensorFlow Datasets을 사용해 CNN(Convolutional Neural Network) 구조의 모델을 훈련
- 모델을 통해 분류할 수 있는 꽃의 종류
  - 장미 (Class 0)
  - 해바라기 (Class 1)
  - 튤립 (Class 2)
  - 민들레 (Class 3)
  - 데이지 (Class 4)

2. InferenceService 파일 작성

```
# InferenceService 배포
kubectl apply -f models/tf-flower.yaml -n kserve-test

# InferenceService 및 Pod 상태 확인
kubectl get inferenceservice -n kserve-test
kubectl get pods -n kserve-test
```

```
username@myserver02:~$ kubectl get isvc -n kserve-test
NAME        URL                                          READY   PREV   LATEST   LATESTREADYREVISION          AGE
tf-flower   http://tf-flower.kserve-test.example.com   True           100      tf-flower-predictor-00001   94s
username@myserver02:~$ kubectl get pods -n kserve-test
NAME                                                  READY   STATUS    RESTARTS   AGE
tf-flower-predictor-00001-deployment-868f4f56b-fdmrn   2/2     Running   0          85s
```

3. Inference Input 파일 작성

- 모델의 추론 형식에 맞추어 Input JSON 파일 작성


```
{
    "instances":[
        {
            "image_bytes":{
                "b64":"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRofHh0aHBwgJC4nICIsIxwc
                2wBDAQkJCQwLDBgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIy
                8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/
                8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhE
                3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKzt
                8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/
                8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhc
                nN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqs
                9oADAMBAAIRAxEAPwC9A42ir9vA0nOOKxYJhkDqe1bNvO0ZA
                75qzDcDAz0qfhl55BqxGE1pCzZwVPt0qJ7MgZQbh7da1Z7br
                +tAPXpTJ4ipyBTVYqwYHBFTezA1ivHNRsuRU1tOlymOBIOo9
                xIM81AODzUtjGzHMfvVRcl6mmOMio4V3PSAtwjBUd60l
```


4. Inference 요청

```
curl -v -H "Host: ${SERVICE_HOSTNAME}" \
      http://${INGRESS_HOST}:${INGRESS_PORT}/v1/models/$MODEL_NAME:predict \
      -d $INPUT_PATH
```

- INGRESS_HOST
  - 클러스터에서 노드의 IP - 10.0.2.22
- INGRESS_PORT
  - Istio IngressGateway 서비스의 포트 - 31903
- MODEL_NAME
  - inferenceservice에서 지정한 모델 이름 - tf-flower
- INPUT_PATH
  - Input 값을 담은 JSON 파일 - @inferences/tf-flower-input.json
- SERVICE_HOSTNAME
  - inferenceservice가 배포한 모델의 엔드포인트 - http://tf-flower.kserve-test.example.com

4. Inference 요청

```
*   Trying 10.0.2.22:31903...
* Connected to 10.0.2.22 (10.0.2.22) port 31903 (#0)
> POST /v1/models/tf-flower:predict HTTP/1.1
> Host: tf-flower.kserve-test.example.com
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Length: 16201
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< content-length: 222
< content-type: application/json
< date: Mon, 21 Oct 2024 13:06:41 GMT
< x-envoy-upstream-service-time: 359
< server: istio-envoy
<
{
    "predictions": [
        {
            "scores": [0.999114931, 9.20989623e-05, 0.000136786344, 0.000337257865, 0.000300533167, 1.84813962e-05],
            "prediction": 0,
            "key": "   1"
        }
    ]
* Connection #0 to host 10.0.2.22 left intact
```

인덱스 0의 score가 0.99로 가장 높으므로, 요청을 보낸 이미지를 장미(Class 0)라고 추론한 것을 볼 수 있음

5. Canary 배포

```yaml
apiVersion: "serving.kserve.io/v1beta1"
kind: "InferenceService"
metadata:
  name: "tf-flower"
  namespace: "kserve-test"
spec:
  predictor:
    canaryTrafficPercent: 20 # 트래픽의 20%를 새로운 모델로 보냄
    model:
      modelFormat:
        name: "tensorflow"
      storageUri: "gs://kfserving-examples/models/tensorflow/flowers-2"
      resources:
        requests:
          cpu: "500m"
          memory: "1Gi"
```

- 기존 : 100%의 트래픽이 기본 모델로 보내짐
- Canary 배포 : canaryTrafficPercent 필드에서 지정한 값만큼의 트래픽이 새로운 버전의 모델로 보내짐

5. Canary 배포

InferenceService 및 Pod 상태 확인

```
username@myserver02:~$ kubectl get pods -n kserve-test
NAME                                                  READY   STATUS    RESTARTS   AGE
tf-flower-predictor-00001-deployment-868f4f56b-fdmrn  2/2     Running   0          20m
tf-flower-predictor-00002-deployment-7767b7fd77-95b8k  2/2     Running   0          54s
username@myserver02:~$ kubectl get isvc -n kserve-test
NAME        URL                                        READY   PREV   LATEST   PREVROLLEDOUTREVISION       LATESTREADYREVISION         AGE
tf-flower   http://tf-flower.kserve-test.example.com   True    80     20       tf-flower-predictor-00001   tf-flower-predictor-00002   20m
```

트래픽 분배 확인

1. 기존 모델 (tf-flower-predictor-00001)

```
username@myserver02:~$ kubectl logs istio-ingressgateway-5f8b7bf488-qbwt6 -n istio-system | grep tf-flower-predictor-00001
[2024-10-21T13:06:06.959Z] "POST /v1/models/tf-flower:predict HTTP/2" 200 - via_upstream - "-" 16201 222 351 350 "10.0.2.22,192.168.131.33" "curl/7.81.0" "4b8a9600-e242-4846-969d-b16c249716
38" "tf-flower-predictor.kserve-test.svc.cluster.local" "192.168.131.34:8012" outbound|80||tf-flower-predictor-00001.kserve-test.svc.cluster.local 192.168.131.33:54924 192.168.131.33:8081 1
92.168.131.33:41810 - -
[2024-10-21T13:06:11.035Z] "POST /v1/models/tf-flower:predict HTTP/2" 200 - via_upstream - "-" 16201 222 382 382 "10.0.2.22,192.168.131.33" "curl/7.81.0" "0a62ff39-8338-9283-a470-6e7e224695
32" "tf-flower-predictor.kserve-test.svc.cluster.local" "192.168.131.34:8012" outbound|80||tf-flower-predictor-00001.kserve-test.svc.cluster.local 192.168.131.33:54924 192.168.131.33:8081 1
92.168.131.33:41810 - -
[2024-10-21T13:06:12.666Z] "POST /v1/models/tf-flower:predict HTTP/2" 200 - via_upstream - "-" 16201 222 350 349 "10.0.2.22,192.168.131.33" "curl/7.81.0" "687771fd-b9d5-4f15-a71c-c357a8a027
a3" "tf-flower-predictor.kserve-test.svc.cluster.local" "192.168.131.34:8012" outbound|80||tf-flower-predictor-00001.kserve-test.svc.cluster.local 192.168.131.33:54924 192.168.131.33:8081 1
92.168.131.33:41304 - -
[2024-10-21T13:06:13.719Z] "POST /v1/models/tf-flower:predict HTTP/2" 200 - via_upstream - "-" 16201 222 291 290 "10.0.2.22,192.168.131.33" "curl/7.81.0" "72375369-7620-46cb-86d9-59628f56ab
f4" "tf-flower-predictor.kserve-test.svc.cluster.local" "192.168.131.34:8012" outbound|80||tf-flower-predictor-00001.kserve-test.svc.cluster.local 192.168.131.33:54924 192.168.131.33:8081 1
92.168.131.33:41810 - -
[2024-10-21T13:06:14.606Z] "POST /v1/models/tf-flower:predict HTTP/2" 200 - via_upstream - "-" 16201 222 304 304 "10.0.2.22,192.168.131.33" "curl/7.81.0" "c2cfcf7e-3803-4ed7-b019-47967dfb33
1b" "tf-flower-predictor.kserve-test.svc.cluster.local" "192.168.131.34:8012" outbound|80||tf-flower-predictor-00001.kserve-test.svc.cluster.local 192.168.131.33:54924 192.168.131.33:8081 1
92.168.131.33:41810 - -
[2024-10-21T13:06:15.397Z] "POST /v1/models/tf-flower:predict HTTP/2" 200 - via_upstream - "-" 16201 222 334 333 "10.0.2.22,192.168.131.33" "curl/7.81.0" "e4201f0d-8ac4-49cf-933e-905dcab65a
0f" "tf-flower-predictor.kserve-test.svc.cluster.local" "192.168.131.34:8012" outbound|80||tf-flower-predictor-00001.kserve-test.svc.cluster.local 192.168.131.33:54924 192.168.131.33:8081 1
92.168.131.33:41810 - -
```

2. Canary 모델 (tf-flower-predictor-00002)

```
username@myserver02:~$ kubectl logs istio-ingressgateway-5f8b7bf488-qbwt6 -n istio-system | grep tf-flower-predictor-00002
[2024-10-21T13:06:36.998Z] "POST /v1/models/tf-flower:predict HTTP/2" 200 - via_upstream - "-" 16201 222 996 995 "10.0.2.22,192.168.131.33" "curl/7.81.0" "29d9cf36-e992-45fa-804f-97909cde1e
99" "tf-flower-predictor.kserve-test.svc.cluster.local" "192.168.131.34:8012" outbound|80||tf-flower-predictor-00002.kserve-test.svc.cluster.local 192.168.131.33:51286 192.168.131.33:8081 1
92.168.131.33:41304 - -
```

PART 5

결론 및 고찰

KServe ?

- KServe는 활발하게 사용될 것인가?
- 효용성이 있는가?

우리FISA 클라우드 엔지니어링 3기 기술 세미나

감사합니다

THANK YOU

허예은