[1. Kubeflow]

1.1 ML Workflow

1.1.1 ML Workflow란

ML Workflow = 문제 해결을 위해..

- 1) 데이터를 분석/가공
- 2) 모델을 학습 + 최적화
- 3) 모델을 서버에 배포
- 4) 예측

하는 전체적인 과정

2개의 단계로 구성

- 1) **모델 실험 단계** (Experiment Phase)
- 2) **모델 생산 단계** (Production Phase)

1.1.2 모델 실험 단계 (Experiment Phase)

문제 해결을 위해 사용될 모델을 실험하는 단계

- 현재 문제가 ML로 풀 수 있는지 + 어떤 모델 사용할지
- 필요한 데이터 수집+분석
- 적합한 ML framework 선택
- 최초의 모델 코드 작성 & 모델 학습
- 하이퍼파라미터 튜닝

1.1.3 모델 생산 단계 (Production Phase)

실험된 모델을 학습+배포하는 단계

- (실험 단계와 맞추기 위해) "실제" 데이터를 재가공
- "실제" 데이터로 모델 학습
- 모델 배포
- 모델 성능 모니터링 & 튜닝/재학습 여부 결정

1.1.4 ML Workflow 툴

- **Pipeline 툴** 형태
  - 각 연결 단계는 독립적
  - 구조에 따라, 병렬적 수행도 가능

  → 효율적인 구성 만들 수 있음
- examples

    - ex 1) **Apache의 Airflow**
        - 덱(Dag), 트리(Tree), 간트(Gantt), 그래프(Graph) 등 다양한 컴포넌트 지원
        - python을 통해 workflow 작성 가능
    - ex 2) **argo workflow**
        - kubeflow 파이프라인의 전신
        - argoproj : 쿠버네티스 위에 실행되는 오픈소스

  → ML 용으로 개발된 것은 아니나, 일반적인 workflow와 크게 다르지 않기때문에 사용 OK

- Public Cloud에서도 관련 tool 제공
    - AWS의 SageMaker
    - GCP의 AI Platform
    - Azure의 Automated ML
    - 알리바바의 Arena

  → 알고리즘 제공, 모델 개발환경 제공, 하이퍼파라미터 튜닝, 모델 배포/서빙

  (= 완전관리형 ML 서비스 제공)

- ML framework에서도 제공
    - Tensorflow Extended (TFX) : 데이터 수집 ~ 모델 서빙까지

이처럼, ML Workflow를 구성할 수 있는 다양한 서비스 & 오픈소스들이 존재!!

1.2. kubeflow

1.2.1 kubeflow

kubeflow = 쿠버네티스를 사용하는 ML 툴킷

(1) Kubeflow의 시작

= **쿠버네티스**에서 **Tensorflow Extended Pipeline**을 사용하면서부터!

→ ***어떻게하면 쿠버네티스에서 Tensorflow Job을 효율적으로 사용할 수 있을까?*** 에서 시작!

(2) Kubleflow의 목표

- ML Workflow에 필요한 서비스를 만드는 것 (X)
- **각 영역에서 가장 적합한 오픈 소스 시스템을 제공** (O)

→ 새로운 서비스 X, **"기존에 있던 오픈소스들의 묶음"**

1.2.2 kubeflow components on ML workflow

ML workflow에 필요한 component들을 제공!

[1] 모델 실험 단계 (Experimental phase)

- ML 알고리즘을 위한 :
  - **Pytorch, sklearn, TensorFlow, XGBoost**
- 실험을 위한 :
  - **Jupyter Notebook, Fairing, Pipelines**
- 하이퍼파라미터 튜닝을 위한 :
  - **Katib**

[2] 모델 생산 단계 (Production phase)

- 모델 학습을 위한 :
  - **Chainer, MPI, MXNet, Pytorch, TFJob**
- 모델 배포를 위한 :
  - **KFServing, NVIDIA TensorRT, PyTorch, TFServing, Seldon**
- 모델 모니터링을 위한 :
  - **Metadata, Tensorboard**

1.2.3 kubeflow UI

- 컴포넌트들은 각각의 **GUI**를 가짐 (+ **대쉬보드 UI**)
- `kfctl`이라는 **CLI**도 지원

1.2.4 API & SDK

- 각 컴포넌트를 관리할 수 있는 **API**
- 컴포넌트 내에 오브젝트(리소스)를 생성할 수 있는 python **SDK**

→ GUI 뿐만 아니라, 다른 애플리케이션에서도 활용 가능!

1.2.5 kubeflow의 component들

총 7가지의 component들

- 1) Jupyter Notebook
  - Juptyer Hub를 서비스
- 2) Main Dashboard
  - component들의 통합 포털
- 3) Hyperparameter Tuning
  - Katib이라는 하이퍼파라미터 최적화 오픈소스
- 4) Pipeline
- 5) Serving
  - Tensorflow Serving, Seldon Serving, Pytorch Serving, KFWerving
- 6) Training
  - TFJob, PyTorch, MXNetc, MPI
- 7) etc

1.2.6 Kubeflow의 version

1. Stable : 배포된 버전 (1.0)
2. Beta : 1.0으로 가기 위해 작업 중인 버전
3. Alpha : 초기 개발 단계

1.3 쿠버네티스 복습

1.3.3 쿠버네티스 구조

Cluster 형태로 구성

- Cluster = Node의 집합

  (최소 1개의 worker node + 1개의 master node)
- Node = 여러 개의 Container를 실행

  (최소 1개의 Container)

Node

- Master node :
  - worker node & cluster내의 component들을 관리
  - kube-api-server, etcd, kube-scheduler, kube-controller-manager 등을 가짐
- Worker node :
  - 실제로 일 하는 노드 (pod를 호스트함)
  - kube-proxy를 가짐

1.3.4 리소스와 컨트롤러

리소스 ex)

- 파드, 볼륨, 서비스, 네임스페이스

컨트롤러 ex)

- 데몬셋, 스테이트풀 셋, 잡 ..

Pod : 쿠버네티스의 "기본 실행/배포 단위"

- node에서 실행되는 "프로세스"
  - 각 node마다 실행되고 있는 kubelet에 의해 관리
- "최소 하나 이상의 container"로 구성
- pod 내의 container들은 pod의 리소스를 공유
- pod 내의 컨테이너는, 컨테이너 각자의 포트를 통해 통신

Volume : 스토리지

- 각 컨테이너들의 "외장 디스크"
- 컨테이너는 stateless, 따라서 종료하면 이전 데이터 모두 사라짐

  → 보존해야 할 때, Volume을 생성하여 pod에 마운트!

  (pod가 사라져도 보존됨)
- 2개로 나누어짐
  - PV (Persistent Volume) : 볼륨 그 자체
    - provisioning = PV를 만드는 것

      (정적 provisioning & 동적 provisioning)
  - PVC (Persistent Volume Claim) : 사용자가 요청하는 볼륨

Controller : 파드 관리

- 운영에 필요한 스케일링/롤아웃/롤백/롤링 업데이트
- ex) ReplicaSet, Deployment, DaemonSet, StatefulSet, Job
- (1) ReplicaSet
  - 지정된 수의 pod가 항상 실행되도록
- (2) Deployment
  - 가장 기본적인 controller
  - Stateless 애플리케이션 배포할 때 사용
  - ReplicaSet의 상위 개념
    - ReplicaSet : Replica 수 만큼의 파드가 생성
    - Deployment : 이러한 ReplicaSet을 생성
- (3) Daemon Set
  - 데몬 프로세스로 띄워놔야 할 앱이 있을 경우
  - ex) 로그 수집기, 모니터링 관련 프로세스
- (4) Stateful Set
  - 상태를 가지고 있는 pod를 관리
  - 삭제는 역순으로
  - 이름 뒤에는 순서를 나타내는 n(0~n)이 붙음
- (5) Job
  - 1개 이상의 pod가 요청된 작업을 실행한 후, 잘 종료되었는지 고나리
  - "한번 만" 실행하는 작업에 대한 컨트롤러
- (6) CronJob
  - 잡을 주기적으로 실행할 수 있는 컨트롤러
- (7) Service
  - 생성한 파드를 외부에서 접근할 수 있게 해주는 리소스
  - pod의 엔드포인트 제공
  - 4종류 (ClusterIP, NodePort, LoadBalancer, ExternalName)

Namespace : 논리적으로 분리된 작업 그룹

- 작업 그룹은 쿠버네티스 리소스를 가짐 (= 리소스 그룹)
- 권한 설정을 통해 공유 & 격리 가능
- 쿠버네티스 설치 시, 생성되는 네임스페이스
    - **default** : 기본 네임스페이스
    - **kube-system** : 쿠버네티스 관리용 파드, 설정이 있는 네임스페이스
    - **kube-public** : 클러스터 내의 "모든" 사용자가 접근 가능한 네임스페이스
    - **kube-node-lease** : 1.13이후 추가된 네임스페이스

1.3.5 오브젝트 템플릿

사용자가 오브젝트(리소스)를 다루는 2가지 방법

- 1) kube-api-server에 **API 리퀘스트** (with JSON 형태)
- 2) kubectl이라는 **CLI** (with yaml 형태)

YAML 형식은 4개의 필드로 구성

- 1) apiversion
- 2) kind
- 3) metadata
- 4) spec

(각각의 리소스 별, 사용하는 하위 field는 다름. 이를 알고프면 `kubectl explain [리소스명]`)

YAML 대신, kubectl 명령 뒤에 인수로 넣어 작업도 가능하나, **YAML 형식 추천!**

1.3.6 label, selector, annotation

label & annotation

- 모두 key=value 형식으로 표현
- 차이점
    - label = 특정 리소스를 "선택하기 위해"
    - annotation = 특정 리소스의 "주석.설명"

selector

- label 기준으로 리소스 선택

NodeSelector

- 특정 label을 가진 node를 선택

1.3.7 Ingress

- **클러스터 외부**에서 들어오는 **요청을 처리하는 규칙**
- 서비스 vs Ingress
  - 서비스 : L4 레이어 (호스트명 라우팅, URL path 라우팅 불가)
  - Ingress : L7 레이어
- Ingress = 규칙 명세가 모인 오브젝트. 그 자체로 실행 불가

  → Ingress Controller 필요

  (ex. nginx ingress controller)

1.3.8 ConfigMap

- 환경설정을 저장하는 리소스
- 컨테이너 내의 애플리케이션에서 사용하는 환경변수를 분리하여 별도로 관리해야!

1.3.9 Secret

- 환경변수에는 DB 비번, 사용자 비번, OAuth Token 같은 정보가 있을 수도

  → ConfigMap이 아닌, **Secret**이라는 별도의 리소스를 통해 관리(보호)
- Secret의 구분
  - 1) 내장 시크릿
    - 쿠버네티스 API에 접근 시, Service Account가 사용하는 시크릿
  - 2) 사용자 정의 시크릿
    - 애플리케이션이 사용

1.4 Kubeflow 설치

1.4.1 설치 조건

2가지가 준비되어 있어야!

- **1) Kubernetes**
  - Public cloud의 것을 사용해도 OK .... GCP의 GKE, AWS의 EKS)
- **2) Kustomize**
  - *쿠버네티스 오브젝트의 배포를 도와주는 툴*

1.4.2 kubernetes 설치

- 생략

1.4.3 Private Docker Registry

- 예저 실행 위해, 도커 이미지 저장할 Private docker registry 설치해야!

1.4.4 k9s

- 쿠버네티스 사용을 위해서는 `kubectl`을 사용했었음

  (but, CLI 특성 상, 모니터링 등의 작업에서 편의성 떨어짐)
- 이를 위해 `k9s`라는 툴을 사용

1.4.5 kfctl

- kubeflow 컴포넌트를 배포/관리하기 위한 CLI 툴

  (`kubectl`과 비슷한 역할)

1.4.6 배포 플랫폼

kubeflow는 **배포 플랫폼**에 따라 배포 설정 파일 제공

크게 2가지 경우

- 1) 이미 존재하는 kubernetes cluser 환경
- 2) public cloud 서비스 환경

(+ 어떠한 인증방식 택하느냐에 따라서도 다름)

1) 이미 존재하는 kubernetes cluser 환경

- (a) standard kubeflow 버전
- (b) Dex for authentication 버전

2) public cloud 서비스 환경

- (a) AWS의 EKS
- (b) Azure의 AKS
- (c) GCP의 GKE

1.4.7 standard kubeflow 설치

(standard kubeflow 버전이든, Dex for authentication 버전이든, 동일한 순서)

1. 설치에 필요한 환경 변수 설정

```
$ export KF_NAME=handson-kubeflow
$ export BASE_DIR=/home/${USER}
$ export KF_DIR=${BASE_DIR}/${KF_NAME}

$ export CONFIG_URI="https://raw.githubusercontent.com/kubeflow/manifests/v1.0-branch/kfdef/kfctl_k8s_istio.v1.0.0.yaml"
```

2. **kustomize 패키지** build하기
    - 설치 폴더에 kustomize라는 폴더가 생성될 것
    - 로컬 버전의 배포 템플릿도 생성

```
$ mkdir -p ${KF_DIR}
$ cd ${KF_DIR}
$ kfctl build -V -f ${CONFIG_URI}
```

3. `kfctl` 을 사용하여, 위의 패키지 실행
    - ***Applied the configuration Sucessfully!***
    - 3개의 namespace가 생성될 것
        - kubeflow
        - knative-serving
        - istio-system

```
$ export CONFIG_FILE=${KF_DIR}/kfctl_k8s_istio.v1.0.0.yaml
$ kfctl apply -V -f ${CONFIG_FILE}
```

4. `k9s` 를 이용하여 pod들의 status 확인하기
    - 모든 pod들이 정상적으로 running되면, kubeflow 배포는 완료!
    - 이제 kubeflow 대쉬보드에 접속 가능

      (istio-system의 istio-ingressgateway 서비스 통해서!)

      (`http://{서버주소}:31380`)

5. 대쉬보드 접속
    - 사용할 namespace 설정 (`seunghan96`)
    - 그러면, 해당 namespace 기반의 대쉬보드로 이동될 것!

설치 완료!!

1.4.8 Dex 버전 설치

- 1.4.7과 동일하나, "배포 패키지 파일만 다를 뿐"
- 최초 로그인 정보 :
  - id) admin@kubeflow.org
  - pw) 12341234

1.4.9 프로파일

- v0.6 버전 이전 : 다중 사용자에 대한 지원 X
- v0.6 버전 이후 : 다중 사용자에 대한 지원 O
  - **프로파일(Profile)**이라는 kubeflow object를 제공함으로써!

프로파일(Profile)

- 프로파일명과 같은 "namespace의 리소스"와 "kubernetes 리소스"의 모음
- 프로파일 내에 속한 사용자는, 이 리소스 가능!
- 프로파일의 소유자는 다른 사용자에게 자신의 프로파일 조회/수정권한 부여 가능!
- 안전하진 않기 때문에, 리소스에 대한 별도의 보안관리는 필요

프로파일 탬플릿

- 프로파일 탬플릿을 관리할 수 있는 사용자는, cluster admin rule을 가지고 있어야

```
apiVersion : kubeflow.org/v1beta1
kind : Profile
metadata :
...
spec :
...
EOF
```

v0.62 버전 이후로 처음 로그인 시, **프로파일 자동 생성기능** 제공

생성되고 나면, 아래의 리소스가 생성될 것

- 1) 프로파일 이름과 같은 namespace
- 2) `namespaceAdmin`
  - 프로파일 소유자 = namespace의 admin
- 3) `ns-access-istio 서비스롤`
  - namespace 내의 서비스가 istio 라우팅할 수 있는 권한
- 4) `owner-biding-istio 서비스롤 바인딩`
  - ns-access-istio를 프로파일 소유자에 바인딩
- 5) `default-editor, default-viewer 서비스어카운트`
  - 사용자가 생성한 리소스에 대한 조회/접근 권한

1.4.10 삭제

최초에 설치했던 config 파일로 진행

```
$ cd ${DF_DIR}
$ kfctl delete -f ${CONFIG_FILE}
```