



추가 공부

이미 애저 학생용을 사용해서 계정이 활성화되지 않는 분들을 위한 추가 실습입니다.

물론 과제로 해주시면 좋지만 부담을 드리지 않기 위해 그냥 심심할 때...에.. 한번 해보셔요

이번 실습은 지난 3주차 베이스세션때 진행했던 도커+쿠버네티스 실습을 토대로 ML 모델을 컨테이너화하고, 쿠버네티스로 배포한 다음 깃허브와 연동하여 cicd 파이프라인을 만드는 실습입니다.

📌 Step 1: 프로젝트 디렉토리 및 파일 생성

```
bash
복사
mkdir mlops-project && cd mlops-project
mkdir app
touch app/main.py app/model.pkl requirements.txt Dockerfile
```

✅ 이 명령어가 하는 일

- `mlops-project` 라는 새로운 디렉토리를 생성하고 이동
- `app/` 디렉토리를 생성
- `app/main.py` (FastAPI 서버 코드) 파일 생성
- `app/model.pkl` (저장된 ML 모델 파일) 생성
- `requirements.txt` (필요한 Python 패키지 목록) 생성

- `Dockerfile` (컨테이너를 위한 설정) 생성

▼ touch 명령어에서 오류가 난다면?

touch 명령어가 안 먹을 경우

Windows에서 빈 파일 생성하기

방법 1: `echo` 명령어 사용

```
bash
복사
echo. > app\main.py
echo. > app\model.pkl
echo. > requirements.txt
echo. > Dockerfile
```

✅ 빈 파일이 정상적으로 생성됩니다!

#프로젝트 파일 구조

생성한 폴더명/

```
| — app/
|   | — main.py
|   | — model.pkl # 모델 저장 위치
| — train_model.py # 모델을 학습하고 model.pkl을 생성하는 파일
| — requirements.txt
| — Dockerfile
```

2 간단한 ML 모델 저장 (`app/model.pkl`)

우리가 사용할 **Scikit-learn** 기반 간단한 모델을 만들어서 `.pkl` 파일로 저장하겠습니다.

✅ `train_model.py` (모델 생성 및 저장)

- **vscode**에서 해당 코드를 미리 만든 디렉토리 폴더에 저장하시면 됩니다.

```
python
복사
import joblib
import numpy as np
from sklearn.linear_model import LogisticRegression

# 더미 학습 데이터
X = np.array([[0], [1], [2], [3], [4], [5]])
y = np.array([0, 0, 0, 1, 1, 1])

# 간단한 로지스틱 회귀 모델 학습
model = LogisticRegression()
model.fit(X, y)

# 모델 저장
joblib.dump(model, "app/model.pkl")
print("✅ 모델이 저장되었습니다!")
```

실행해서 모델을 저장하세요. 저는 VScode 터미널에서 확인작업을 거쳤습니다.

```
PS C:\Users\user> d:
PS D:\> cd mlops
PS D:\mlops> python train_model.py
✅ 모델이 저장되었습니다!
PS D:\mlops>

PS D:\mlops> ls app

디렉터리: D:\mlops\app

Mode                LastWriteTime         Length Name
----                -
-a----          2025-02-25 오후 11:24             3 main.py
-a----          2025-02-25 오후 11:35           831 model.pkl
```

pkl파일이 잘 생성된 것을 볼 수 있습니다. 우리는 모델을 "영구적으로 저장"하고, 필요한 때에 언제든지 "불러와서 사용할 수 있는" 파일을 생성했습니다.

3 FastAPI로 ML 모델 API 만들기 (`app/main.py`)

```
#main.py

from fastapi import FastAPI
import joblib
import numpy as np
import pandas as pd
import os

app = FastAPI()

# 절대 경로로 모델 로드
model_path = os.path.join(os.path.dirname(__file__), "model.pkl")
model = joblib.load(model_path)

@app.get("/")
def home():
    return {"message": "ML 모델 API 입니다."}

@app.post("/predict/")
def predict(data: dict):
    df = pd.DataFrame([data])
    prediction = model.predict(df)
    return {"prediction": int(prediction[0])}
```

✅ API 테스트

```
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

➡ <http://127.0.0.1:8000/docs> 로 이동해서 테스트 가능!

```
IndentationError: unexpected indent
INFO: Stopping reloader process [24880]
PS D:\mlops> uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
INFO: Will watch for changes in these directories: ['D:\mlops']
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started reloader process [25288] using StatReload
INFO: Started server process [8040]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

```
state.py: print("종료")
message="ML 모델 API입니다")

import uvicorn
import numpy as np
import pandas as pd

app = FastAPI()

# 서킷브레이크 설정
model = joblib.load("app/model.pkl")

@app.get("/")
def read_root():
    return {"message": "Hello World"}

if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8000)
```

File "frozen importlib_bootstrap_external", line 1004, in source_to_code
File "frozen importlib_bootstrap", line 241, in _call_with_frames_removed
File "D:\workspace\ml\app.py", line 1
from fastapi import FastAPI
IndentationError: unexpected indent
INFO: Stopping uvicorn process [24880]
PS D:\workspace\ml\app> uvicorn app:main --host 0.0.0.0 --port 8000 --reload
INFO: Will watch for changes in these directories: (D:\workspace\ml)
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started uvicorn process [2508]
INFO: Started server process [2508]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:5333 - "GET / HTTP/1.1" 200 OK
INFO: 127.0.0.1:5333 - "GET /favicon.ico HTTP/1.1" 404 Not Found

FastAPI 0.1.0 OAS 3.1

/openapi.json

default

- GET / Home
- POST /predict/ Predict

Schemas

- HTTPValidationError > Expand all object
- ValidationError > Expand all object

api 확인 테스트

```
{
  "0": 2
}
```

Parameters Cancel Reset

No parameters

Request body required application/json

```
{
  "0": 2
}
```

Execute Clear

Responses

```

2 import joblib
3 import numpy as np
4 import pandas as pd
5
6 app = FastAPI()
7
8 # 저장된 모델 로드
9 model = joblib.load("app/model.pkl")
10
11 @app.get("/")

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS uvicorn + - 🗑 ...

```

INFO: Stopping reloader process [24880]
PS D:\mlops> uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
INFO: Will watch for changes in these directories: ['D:\mlops']
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started reloader process [25288] using StatReload
INFO: Started server process [8040]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:6333 - "GET / HTTP/1.1" 200 OK
INFO: 127.0.0.1:6333 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:6334 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:6334 - "GET /openapi.json HTTP/1.1" 200 OK
C:\Users\user\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:486: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
INFO: 127.0.0.1:6399 - "POST /predict/ HTTP/1.1" 200 OK

```

Responses

Curl

```
curl -X 'POST' \
'http://127.0.0.1:8000/predict/' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "0": 2
}'
```

Request URL

<http://127.0.0.1:8000/predict/>

Server response

Code	Details
200	<p>Response body</p> <pre>{ "prediction": 0 }</pre> <p>📄 Download</p> <p>Response headers</p> <pre>content-length: 16 content-type: application/json date: Tue, 25 Feb 2025 14:42:41 GMT server: uvicorn</pre>

Responses

모델 api화에 성공했습니다. 여러분도 prediction : 0이 나오면 제대로 된 것 맞습니다.

4 Docker 컨테이너화

✓ requirements.txt

requirements.txt

✓ Dockerfile

```
# Python 3.9 환경 사용
FROM python:3.9

# 작업 디렉토리 설정
WORKDIR /app

# 필요한 패키지 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 코드 복사
COPY app /app

# FastAPI 실행
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

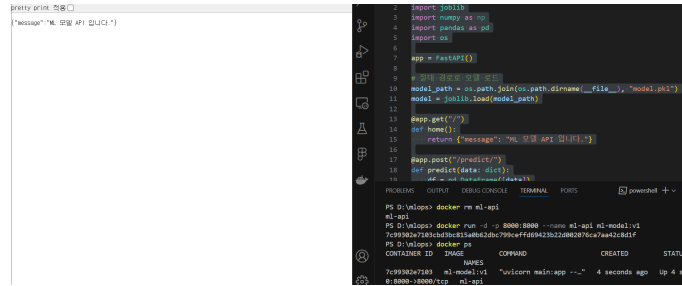
✓ Docker 이미지 빌드 및 실행

```
docker build -t ml-model:v1 .
docker run -d -p 8000:8000 --name ml-api ml-model:v1
```

```
=> exporting to image 29.6s
=> => exporting layers 23.4s
=> => exporting manifest sha256:99cd37cfa47421ed4ee8bc4559d97314282d 0.0s
=> => exporting config sha256:1bd558e6cc0b245aa513d7d274ad5dd799ccc6 0.0s
=> => exporting attestation manifest sha256:c80c6eae70496bb703ae320 0.0s
=> => exporting manifest list sha256:6ef0dfb1e78882a8c2a3a422a457dc 0.0s
=> => naming to docker.io/library/ml-model:v1 0.0s
=> => unpacking to docker.io/library/ml-model:v1 6.0s
```

```
PS D:\mlops> docker run -d -p 8000:8000 --name ml-api ml-model:v1
7c99382e7103cbd3bc815a0b62dbc799ceffd69423b22d002076ca7aa42c8d1f
PS D:\mlops> docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                    NAMES
7c99382e7103   ml-model:v1   "uvicorn main:app --..." 4 seconds ago  Up 4 seconds  0.0.0.0:8000->8000/tcp   ml-api
```

반드시 컨테이너가 실행되고 있는지를 꼭꼭 확인하신 다음 진행하세요



✓ API 테스트 - cmd에서 하는 걸 추천

curl -X 'POST' 'http://127.0.0.1:8000/predict/' -H 'Content-Type: application/json' -d '{"0": 2}'

```
D:\ml>curl -X POST "http://127.0.0.1:8000/predict/" -H "Content-Type: application/json" -d '{"0": 2}'
{"prediction": 0}
D:\ml>
```

➡ {"prediction": 0} 이런 결과가 나오면 정상 작동!

✓ Step 2: Kubernetes로 배포

이제 Docker로 만든 컨테이너를 **Kubernetes** 환경에 배포할 거예요.

✓ ml-deployment.yml (Kubernetes 배포 파일)

```
# 쿠버네티스 배포용
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-api
spec:
  replicas: 2 # 두 개의 컨테이너로 운영
  selector:
    matchLabels:
      app: ml-api
  template:
    metadata:
```



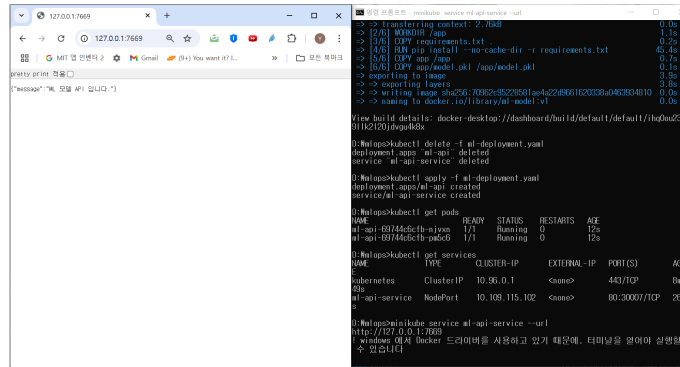
```
labels:
  app: ml-api
spec:
  containers:
    - name: ml-api
      image: ml-model:v1 # 우리가 만든 Docker 이미지
      ports:
        - containerPort: 8000
  ---
apiVersion: v1
kind: Service
metadata:
  name: ml-api-service
spec:
  type: NodePort
  selector:
    app: ml-api
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
      nodePort: 30007 # 클러스터 외부에서 접근 가능
```

✅ Minikube를 이용한 배포

```
kubectl apply -f ml-deployment.yml
kubectl get pods
kubectl get services
```

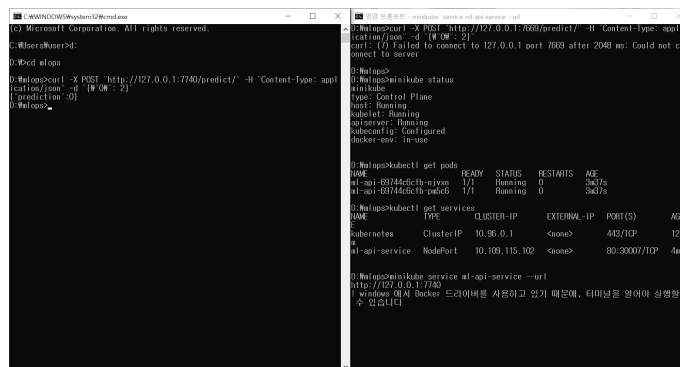
✅ API 확인

```
minikube service ml-api-service --url
```



```
0:~$ docker build -t dashboar0/default/default/fig0na23 0:~$ docker run -d --name=fig0na23 dashboar0/default/default/fig0na23
0:~$ kubectl delete -f ml-deployment.yaml
deployment.apps/ml-api deleted
service/ml-api-service deleted
0:~$ kubectl apply -f ml-deployment.yaml
deployment.apps/ml-api created
service/ml-api-service created
0:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
ml-api-69744dcfb-njwn 1/1 Running 0 12s
ml-api-69744dcfb-pdc6 1/1 Running 0 12s
0:~$ kubectl get services
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 8m
ml-api-service NodePort 10.109.115.102 <none> 80:30007/TCP 2s
0:~$ kubectl describe service ml-api-service -o json
apiVersion: v1
kind: Service
metadata:
  name: ml-api-service
  namespace: default
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app: ml-api
```

➡ 이 URL로 접속해서 FastAPI가 실행되는지 확인!(오른쪽 명령창 밑에 url 보이시죠?)



```
0:~$ curl -X POST http://127.0.0.1:7740/predict/ -H 'Content-Type: application/json' -d '{"OW": 2}'
curl: (0) failed to connect to 127.0.0.1 port 7740 after 2040 ms: Could not connect to server
0:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
ml-api-69744dcfb-njwn 1/1 Running 0 363s
ml-api-69744dcfb-pdc6 1/1 Running 0 363s
0:~$ kubectl get services
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 12m
ml-api-service NodePort 10.109.115.102 <none> 80:30007/TCP 4m
0:~$ kubectl describe service ml-api-service -o json
apiVersion: v1
kind: Service
metadata:
  name: ml-api-service
  namespace: default
spec:
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app: ml-api
```

➡ { "prediction": 0 } 이런 결과가 나오면 정상 작동!

✓ Step 3: CI/CD (GitHub Actions)

이제 모델이 업데이트되면 자동으로 배포되도록 **GitHub Actions + Docker Hub + Kubernetes** 를 활용할 거예요.

✓ CI/CD 흐름

1. **GitHub**에 코드 푸시하면(이때, 로그인을 기존처럼 cmd에 id, pw 처럼 하는 거에서 개인 키였나? 그걸 받아야하는 것으로 변경되었습니다)
2. **GitHub Actions**가 실행되어
3. **Docker Hub**에 최신 ML 모델 컨테이너 푸시
4. **Kubernetes**에서 Rolling Update 적용

✓ GitHub Actions 설정 ([.github/workflows/deploy.yml](https://github.com/actions/workflows/deploy.yml))

name: Deploy ML Model

on:

push:

branches:

- main

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- name: 레포지토리 체크아웃

uses: actions/checkout@v3

- name: Docker 로그인

run: echo "\${{ secrets.DOCKER_PASSWORD }}" | docker login -u "\${{ secrets.DOCKER_USERNAME }}" --password-stdin

- name: Docker 이미지 빌드 및 푸시

run: |

docker build -t my-docker-hub/ml-model:\${{ github.sha }} .

docker tag my-docker-hub/ml-model:\${{ github.sha }} my-docker-hub/ml

-model:latest

docker push my-docker-hub/ml-model:\${{ github.sha }}

docker push my-docker-hub/ml-model:latest

- name: Kubernetes Rolling Update

run: |

kubectl set image deployment/ml-api ml-api=my-docker-hub/ml-model:\${{ github.sha }}

kubectl rollout status deployment/ml-api

✓ GitHub Secrets 설정

1. `DOCKER_USERNAME` → Docker Hub 아이디
2. `DOCKER_PASSWORD` → Docker Hub 비밀번호

✓ Step 4: 모델 업데이트 및 배포 자동화

모델을 업데이트하고 푸시하면, CI/CD가 실행되어 **Kubernetes에 자동 배포**됩니다.

✓ 새로운 모델 훈련 후 저장

```
python train_model.py
```

✓ GitHub에 푸시

```
git add .  
git commit -m "새로운 모델 업데이트"  
git push origin main
```

➡ **GitHub Actions 실행 → Docker Hub → Kubernetes 자동 배포 🚀**

🎯 정리

✓ ML 모델을 FastAPI 컨테이너로 패키징

✓ Docker 컨테이너 빌드 및 실행

✓ Kubernetes에 ML API 배포

▼ 기존에 ds나 ai engineer 업무만 진행했던 것과의 차이점

맞아요! 이전에는 로컬에서만 모델을 돌리고, 그 모델을 내가 직접 호출하거나, 그냥 결과만 받는 방식이었다면, 지금은 다른 사람들이 내가 만든 모델을 사용할 수 있게 되었고, 모델이 자동으로 학습하고 배포되는 시스템을 만들었다는 점에서 큰 차이가 있어요.

핵심 포인트

1. 모델을 다른 사람들이 사용할 수 있게 했다

- 로컬에서 돌리던 모델을 **FastAPI**로 API로 만들었고, 이를 **Docker**와 **Kubernetes**를 사용하여 클라우드 환경에서 **서비스 형태로 배포**했기 때문에, 이제는 **다른 사용자들도 API를 통해** 이 모델을 쉽게 사용할 수 있게 되었어요.

2. 자동 학습과 성능 개선

- **CI/CD 파이프라인**을 구축함으로써, 모델을 업데이트하거나 개선된 모델을 **자동으로 배포**할 수 있게 됐습니다. 예를 들어, 새로운 데이터로 모델을 재학습시키면, **자동으로 새로운 모델을 클라우드에 배포**하고 어플리케이션에 반영됩니다.
- 또한, **모델 모니터링**과 **성능 저하 시 자동 재훈련** 기능을 추가하면, 모델이 **지속적으로 최신 상태**를 유지하며 성능을 향상시킬 수 있습니다.

3. 도커와 쿠버네티스의 역할

- **Docker:** 모델을 컨테이너화하여, 어디서든 동일한 환경에서 실행할 수 있게 만들어줍니다. 즉, 모델을 다른 서버나 클라우드에서 쉽게 실행할 수 있도록 환경을 패키징해줍니다.
- **Kubernetes:** 모델을 확장성 있게 관리하고, 여러 인스턴스를 운영할 수 있게 해줍니다. 예를 들어, 사용자 요청이 많을 때 **자동으로 서버를 확장**시켜 예측 성능이 떨어지지 않도록 유지하며, **장애 발생 시 자동 복구**도 가능합니다.

요약

- **기존:** 로컬에서 모델을 학습하고 실행하고, **한정된 환경에서만 사용**.
- **현재:** Docker와 Kubernetes를 활용해 모델을 **API로 제공하고, 자동 학습과 배포, 다수의 사용자도 접근 가능**하도록 만든 시스템.
- **결과:** 모델을 클라우드에서 **안정적으로 운영**하면서 **지속적으로 성능 개선**을 할 수 있는 시스템을 구축했어요!

이제 모델을 다른 사람들과 공유하고, 실제 서비스로 운영할 수 있는 환경을 만들었기 때문에, **MLOps**의 핵심인 **지속적인 모델 관리**와 **운영 환경에서의 모델 배포**를 모두 경험한 거죠.