

A photograph of a dense forest. Sunlight filters through the tall, thin trunks of coniferous trees, creating bright highlights and deep shadows. The ground is covered in a mix of green grass and dark, leaf-littered soil. In the background, more trees stand in a misty, hazy light.

2025

# MLOps 심화

들어가며

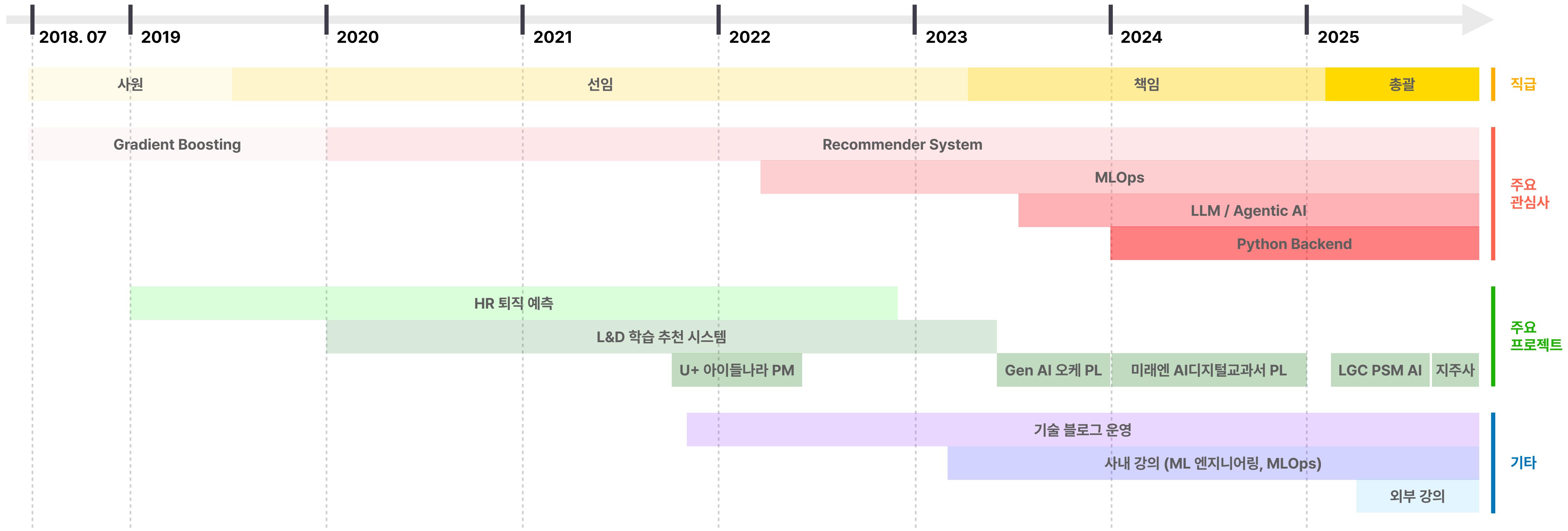
# 반갑습니다



**한재윤** 총괄

AI클라우드사업부 Data Analytics 센터

Enterprise Data사업담당 AI Analytics사업2팀



들어가며

# 시작하기에 앞서

---

## 🔑 본 강의의 핵심

- ▶ 지난 MLOps 강의에서 다루었던 **각 요소와 개발 도구에 대해 더 깊은 내용**을 다룰 예정
- ▶ ML 모델 운영에 있어 **핵심적인 도구(Docker, Airflow)**를 최대한 활용
- ▶ 더욱 많아질 MLOps, LLMOps 관련 프로젝트에서 **다른 직무와의 소통에 도움**이 되도록 하는 것이 목표

## 🚀 핸즈온 실습

- ▶ **실습 위주**로 실제 데이터를 전처리하고 모델을 학습, 배포하는 작업을 자동화하는 연습
- ▶ **각 코드의 중요 부분을 비워놓고** 수강생이 해당 부분을 채우는 방식
- ▶ 되도록 **쉘 환경**을 사용
- ▶ 프로젝트 구성에 대한 **Best Practice** 실습

들어가며

# 강의 일정

	1일차	시수	2일차	시수
09:00 - 09:30	<b>M1. MLOps 개념</b> - MLOps 프로세스 살펴보기	1.0	<b>M3. MLOps 파이프라인 개발 (CT)</b> - 데이터 추출 - 데이터 전처리	1.0
09:30 - 10:00				
10:00 - 10:30	<b>M1. MLOps 개념</b> - 활용 기술 스택 알아보기 - 개발 환경 설정	1.0	<b>M3. MLOps 파이프라인 개발 (CT)</b> - 데이터 전처리 (계속) - 모델 학습/평가 with MLflow	1.0
10:30 - 11:00				
11:00 - 11:30		0.5	<b>M3. MLOps 파이프라인 개발 (CT)</b> - 모델 학습/평가 with MLflow (계속)	0.5
11:30 - 13:00	<b>점심 시간</b> 			
13:00 - 13:30	<b>M2. 활용 도구 익히기</b> - Docker	1.0	<b>M3. MLOps 파이프라인 개발 (CT)</b> - 모델 학습/평가 with MLflow (계속)	1.0
13:30 - 14:00			<b>M4. MLOps 파이프라인 개발 (CD)</b> - 모델 배포(API 개발)	
14:00 - 14:30	<b>M2. 활용 도구 익히기</b> - Docker - SQLAlchemy	1.0	<b>M4. MLOps 파이프라인 개발 (CD)</b> - 모델 배포(API 개발) (계속)	1.0
14:30 - 15:00				
15:00 - 15:30	<b>M2. 활용 도구 익히기</b> - Airflow	1.0	<b>M4. MLOps 파이프라인 개발 (CD)</b> - 지속적 배포 구현	1.0
15:30 - 16:00				
16:00 - 16:30	<b>M2. 활용 도구 익히기</b> - Airflow (계속)	1.0	<b>M4. MLOps 파이프라인 개발 (CD)</b> - 지속적 배포 구현 (계속)	1.0
16:30 - 17:00				
17:00 - 17:30	<b>Wrap-up</b>	0.5	<b>Wrap-up</b>	0.5



# M1. MLOps 개념

1. MLOps 프로세스 살펴보기
2. 활용 기술 스택 알아보기
3. 개발 환경 살펴보기

# **1.0 MLOps 다시 알아보기**

## MLOps 정의하기

$$\text{ML} + \text{DevOps} = \text{MLOps}$$

**DevOps**

소프트웨어 개발 및 운영 생산성을 높이기 위한 **방법론**이면서  
소프트웨어 개발과 인프라 관리 과정에서 전통적인 방식을 사용하는 조직에 비해 제품을 더  
빠르게 진화 및 개선시키는 것을 목표로 하는 **개발 문화**

**MLOps**

ML 모델의 프로덕션 배포와 유지 관리를 자동화하기 위한 **방법론**이면서  
ML 모델의 생산, 배포, 유지 보수를 자동화하고 개선하는 것을 목표로 하는 **개발 문화**



요약하자면 MLOps는 ML 모델의 배포와 유지보수를 자동화하고 개선하기 위한 **방법론**이면서 **개발 문화**

# DevOps와 MLOps의 간단한 비교

---

## DevOps

---



소프트웨어 개발자 간의 소통과 협업을 강조하는 개발 문화

### CI (Continuous Integration)

개발자를 위한 자동화 프로세스인 지속적인 통합  
(코드 충돌 문제 해결)

### CD (Continuous Deployment)

지속적인 서비스 제공 및 지속적인 배포

## MLOps

---



DS와 엔지니어 간의 소통과 협업을 강조하는 ML 엔지니어링 문화

### CI (Continuous Integration)

데이터, 데이터 스키마, 모델에 대해 테스트하고 검증을 추가로 수행

### CD (Continuous Deployment)

모델 학습 및 예측 서비스를 자동으로 배포하는 시스템

### CT (Continuous Training)

ML 시스템만의 고유 속성으로 모델을 자동으로 재학습시키고 서비스 제공

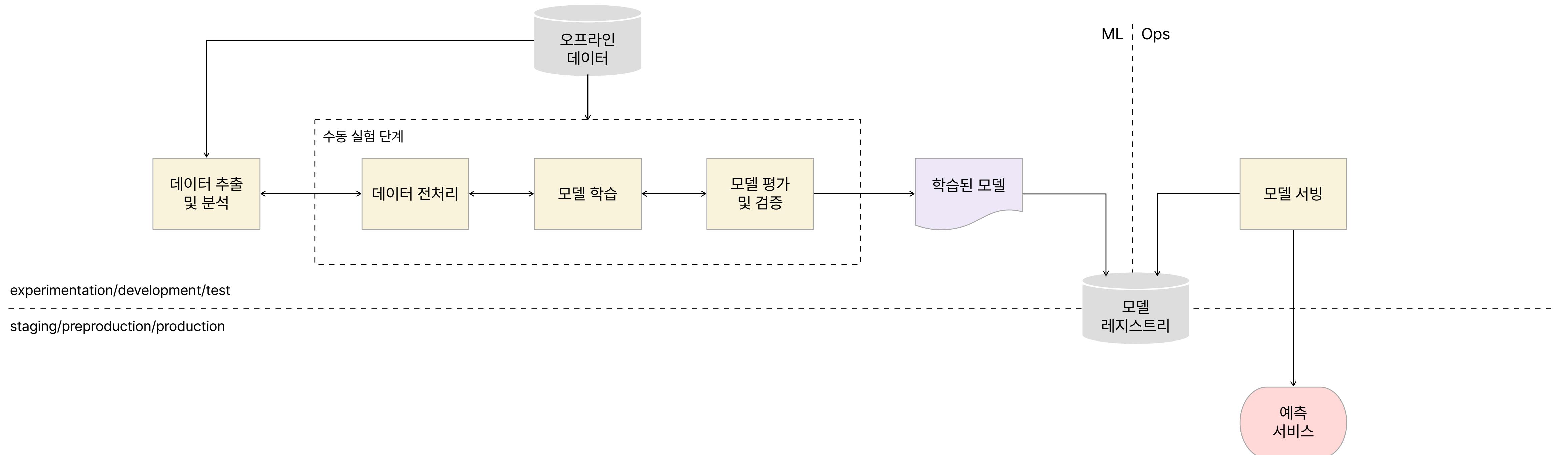
# **1.1 MLOps 프로세스 살펴보기**

## **— Google의 단계별 비교**

## 1.1 MLOps 프로세스 살펴보기

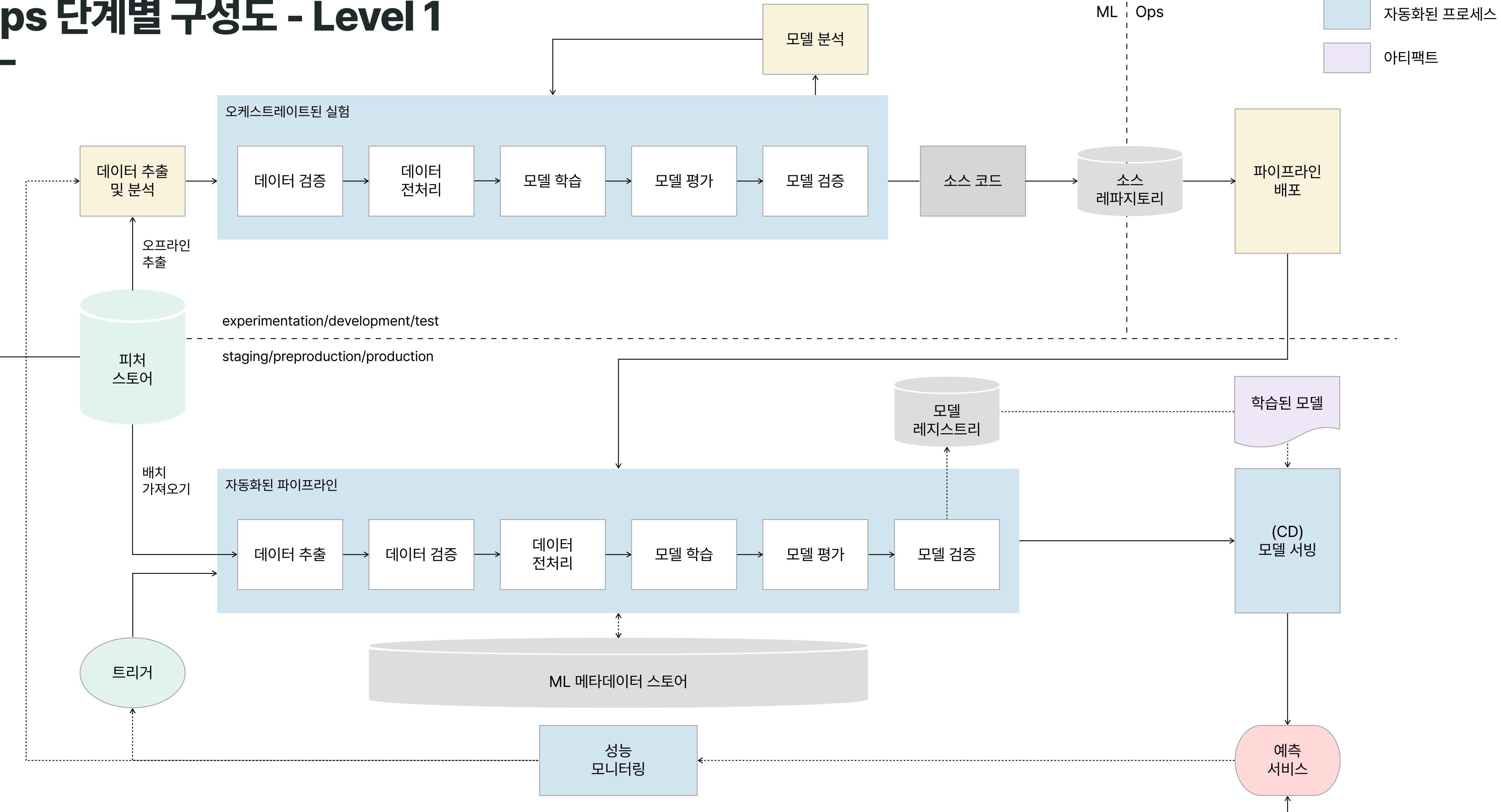
# MLOps 단계별 구성도 - Level 0

- 수동 프로세스
- 자동화된 프로세스
- 아티팩트



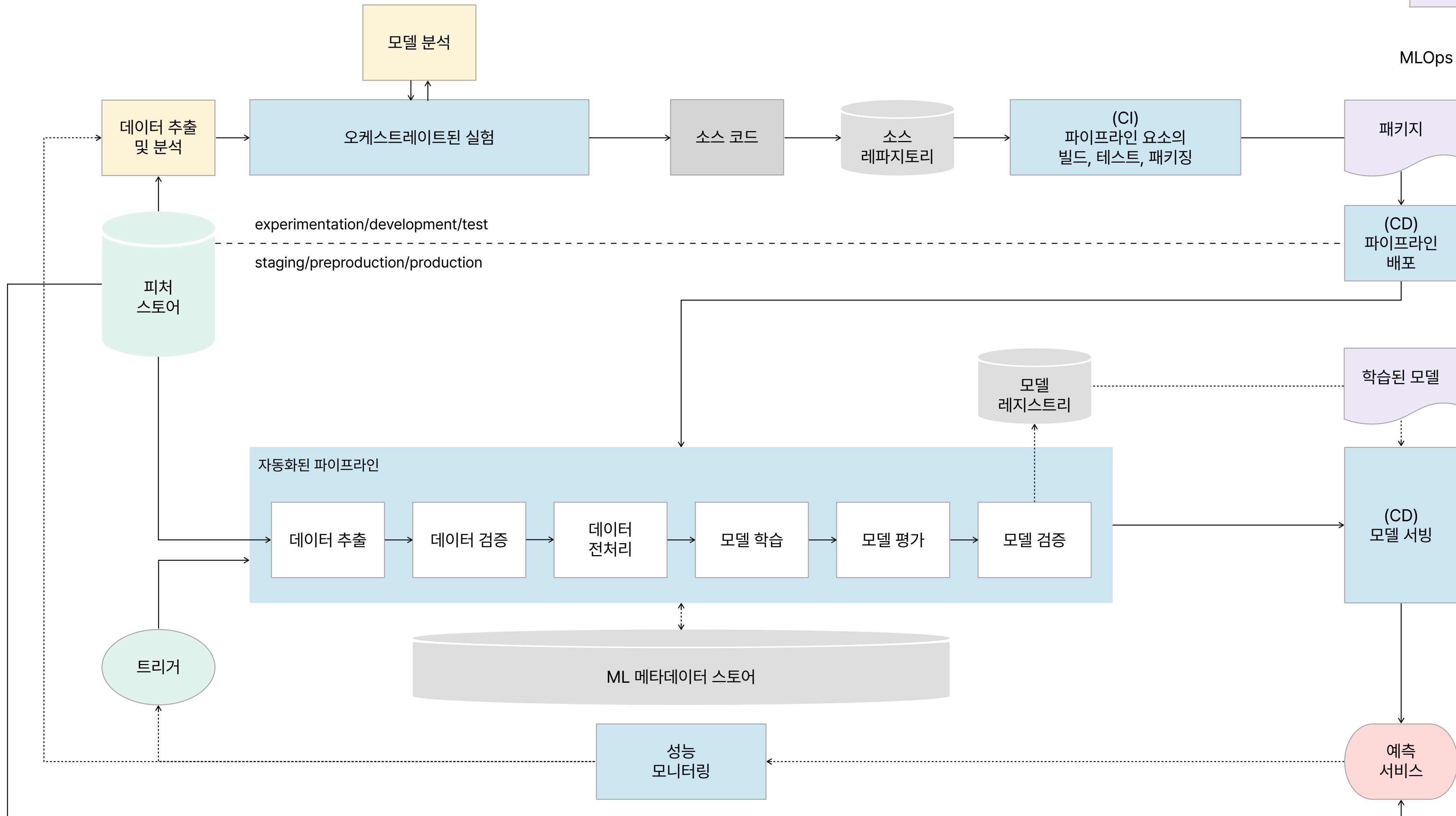
## 1.1 MLOps 프로세스 살펴보기

# MLOps 단계별 구성도 - Level 1



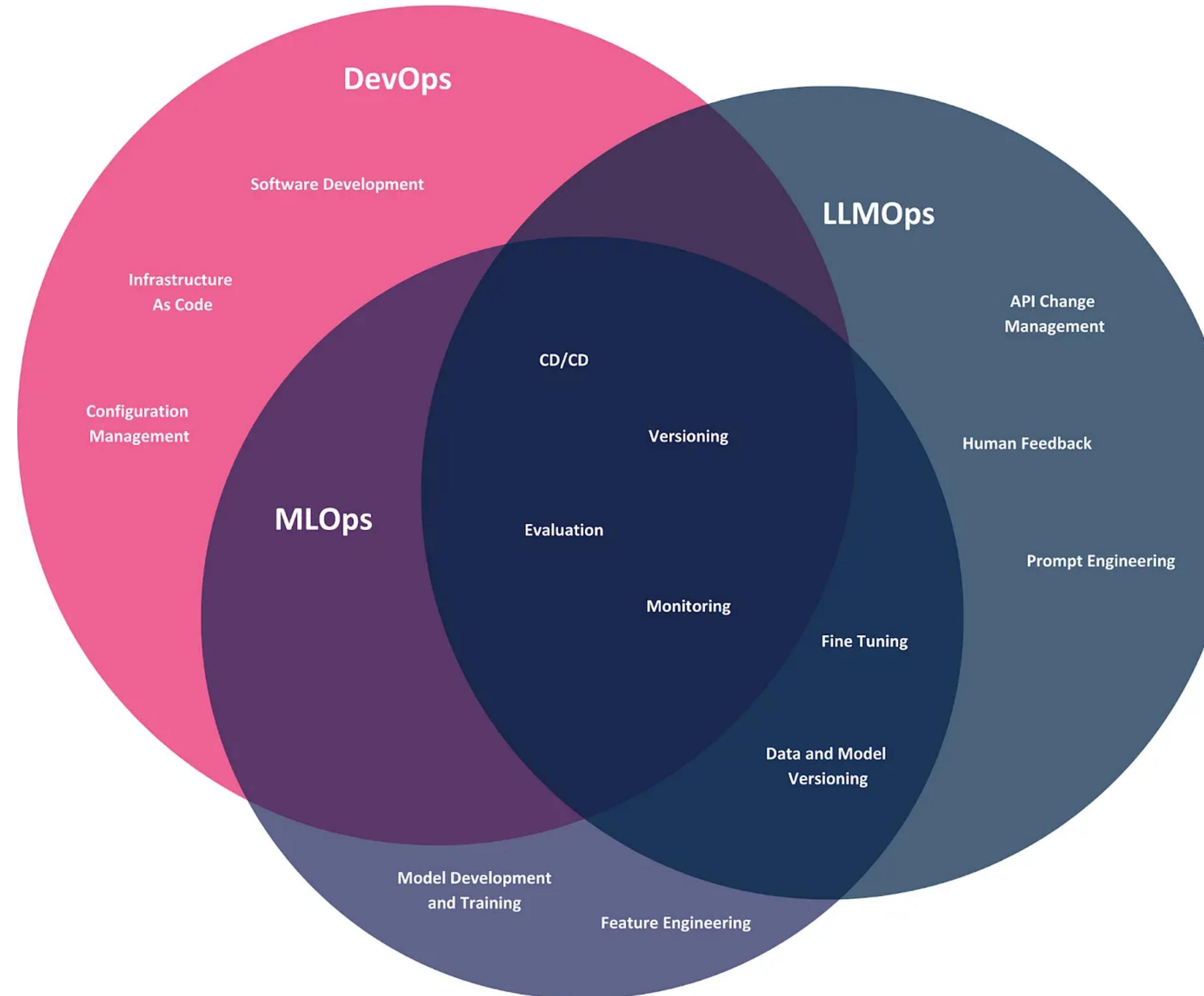
## 1.1 MLOps 프로세스 살펴보기

# MLOps 단계별 구성도 - Level 2



## 1.0 MLOps 다시 알아보기

# MLOps와 LLMOps의 차이



### 🤖 모델 개발의 주체

- ML은 사용자가 직접 개발
- LLM은 일반적으로 사전 학습된 **Foundation Model** 사용

### 📝 프롬프트 엔지니어링

- LLM 애플리케이션 품질은 **프롬프트에 좌우됨**
- 프롬프트 변화에 대한 버전 관리 필수

### 🤖 파인튜닝의 역할

- ML에서는 **모델의 하이퍼파라미터**를 파인튜닝하여 성능 향상 시도  
➡️ 파인튜닝을 위한 추가 인프라가 필요 없음
- LLM에서는 **특정 도메인이나 작업에 맞는 답변이 나오도록** 파인튜닝 수행  
➡️ 파인튜닝을 수행하기 위한 **추가 인프라가 반드시 필요**

### 📊 평가 방식

- ML은 기준에 따른 정량적 평가가 쉬움
- LLM은 정량적 평가가 어려워 정성적/질적 평가에 초점

## 1.0 MLOps 다시 알아보기

# MLOps와 LLMOps의 차이

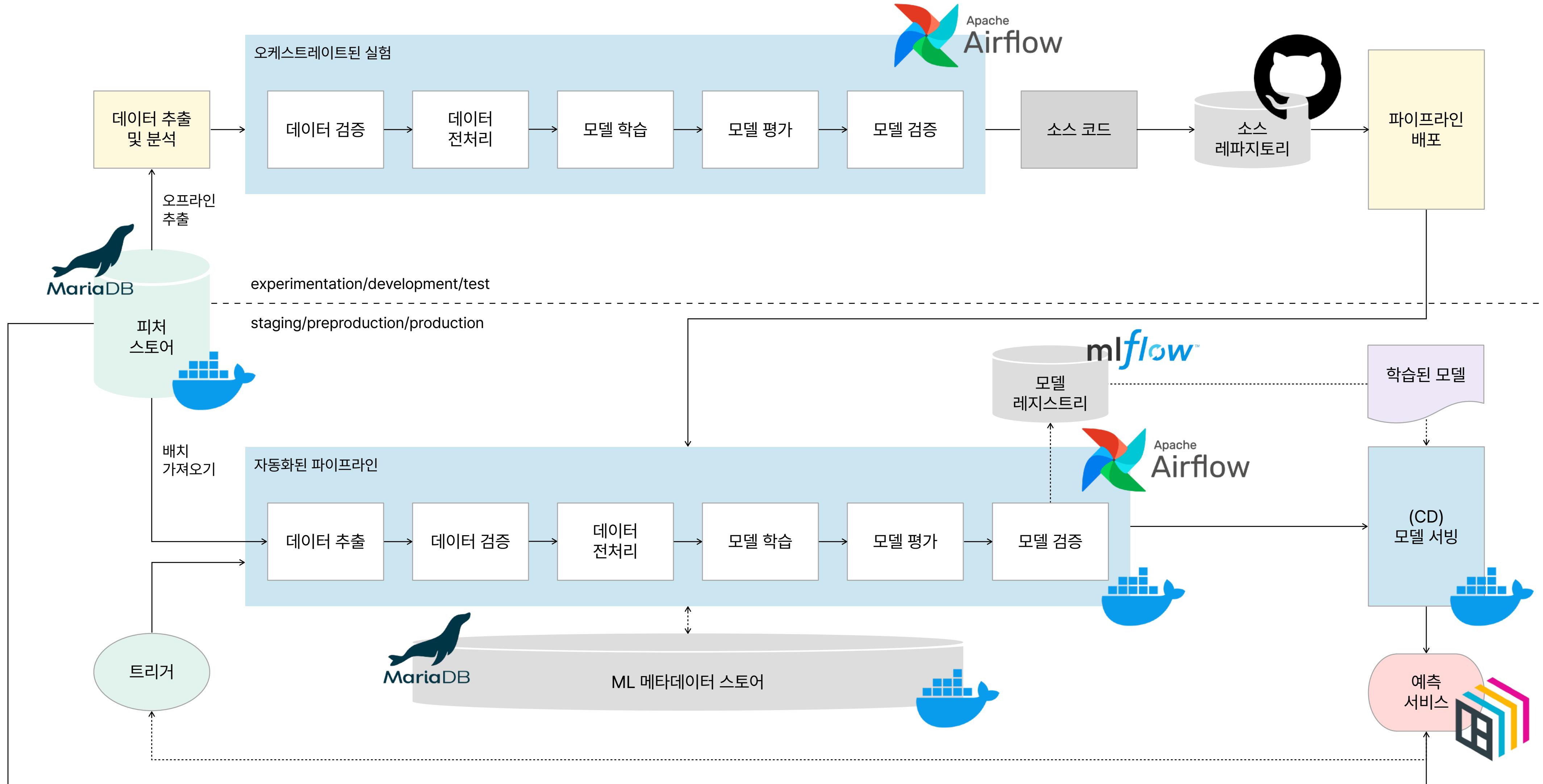
	<b>MLOps</b>	<b>LLM</b> <b>Ops</b>
 모델	<ul style="list-style-type: none"> <li>주로 자체 데이터로 모델을 처음부터 학습</li> <li>피처 엔지니어링이 중요</li> </ul>	<ul style="list-style-type: none"> <li>사전 학습된 <b>Foundation Model</b>을 활용</li> <li>프롬프트 설계 및 관리가 핵심</li> </ul>
 실험 관리	하이퍼파라미터, 코드, 데이터셋 버저닝	<ul style="list-style-type: none"> <li><b>MLOps의 모든 요소 포함</b></li> <li><b>프롬프트</b>, 응답 결과, 파인튜닝 데이터셋</li> </ul>
 인프라	상대적으로 적은 컴퓨팅 자원	<ul style="list-style-type: none"> <li>직접 모델을 서빙하는 경우 <b>GPU 등 컴퓨팅 자원 필요</b></li> <li>이에 따른 <b>효율적인 서빙 및 추론 최적화</b> 필요 (vLLM)</li> </ul>
 모니터링	데이터의 분포 변화, 예측값과 실제값의 차이	<ul style="list-style-type: none"> <li>성능에 대한 판단이 <b>더 복잡하고 주관적</b></li> <li>환각(Hallucination), 유해성, 편향성(Bias) 등 <b>정성적이고 질적인 평가</b> 중요</li> <li>사용자의 프롬프트와 모델의 응답에 대한 지속적인 추적 및 분석 수행</li> </ul>
 비용	모델 학습 및 추론 비용이 거의 들지 않음	<ul style="list-style-type: none"> <li><b>상용 API 사용 시 호출 비용</b> 고려 필요</li> <li>모델이 커질 수록 비용이 일반적으로 증가하여 서빙 비용이 매우 높음</li> </ul>
 고려 요소	데이터 파이프라인, 모델 서빙 프레임워크	<ul style="list-style-type: none"> <li>MLOps의 모든 요소 포함</li> <li>임베딩 관리를 위한 벡터 데이터베이스</li> <li>프롬프트 템플릿 및 체인 관리</li> </ul>

## **1.2 활용 기술 스택 알아보기**

## 1.2 활용 기술 스택 알아보기

# 실습 워크플로우

- 수동 프로세스
- 자동화된 프로세스
- 아티팩트



## 1.2 활용 기술 스택 알아보기

# 활용 기술 스택

---



Docker

- ▶ 컨테이너 기반의 OS 수준의 **가상화 플랫폼** ([자세히 알아보기](#))
- ▶ 운영 환경과 관련된 모든 요소(DB, 학습 파이프라인, 예측 서비스 등)를 모두 **Docker**를 이용해 컨테이너로 띄울 예정



MariaDB  
mariadb

- ▶ 대표적인 오픈소스 RDBMS 중 하나
- ▶ **피처 스토어와 로그 저장소로 활용**하며 Docker를 이용해 띄울 예정



GitHub

- ▶ 소스 코드를 관리하기 위해 사용할 Git 플랫폼
- ▶ 본 실습에선 다루지 않지만 **GitHub Actions**를 이용한 CI도 가능함



Apache Airflow

- ▶ 2014년 에어비엔비에서 개발한 **파이프라인 관리를 위한 오픈소스 워크플로우 관리 플랫폼**
- ▶ 본 실습에서 개발하는 모든 **파이프라인은 Airflow로 오케스트레이션**할 예정



MLflow

- ▶ ML 라이프사이클에서 모델 개발 및 실험 과정을 쉽게 추적할 수 있도록 도와주는 오픈소스 플랫폼
- ▶ **모델 레지스트리와 ML 메타데이터 관리를 위해 사용**



BentoML

- ▶ ML 모델을 서비스로 간단하게 배포하고 관리하기 위한 오픈소스 플랫폼
- ▶ 본 실습에서는 **개발한 모델을 서빙하여 예측 서비스로 띄울 때 FastAPI 대신 사용**할 예정

## **1.3 개발환경 설정**

### 1.3 개발환경 설정

## 저장소 생성

The screenshot shows a GitHub repository page for the user 'otzslayer' named 'advanced-mlops'. The repository is public and has 1 branch and 0 tags. The main branch is 'main'. The repository was created by 'otzslayer' and has 1 commit. The commit message is '최초 커밋'. The files listed are 'api', 'db', 'pipelines', 'utils', '.env', '.gitignore', and 'README.md', all of which were committed 2 hours ago. The 'Code' tab is selected. On the right side, there is an 'About' section with the message 'No description, website, or topics provided.' It also shows 0 stars, 1 watching, and 0 forks. A red box highlights the 'Fork' button, which has 0 forks.

otzslayer / advanced-mlops

Type / to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

advanced-mlops Public

Pin Unwatch 1 Fork 0 Star 0

main 1 Branch 0 Tags

Go to file Add file Code

otzslayer 최초 커밋 099412c · 2 hours ago 1 Commit

File	Commit Message	Time Ago
api	최초 커밋	2 hours ago
db	최초 커밋	2 hours ago
pipelines	최초 커밋	2 hours ago
utils	최초 커밋	2 hours ago
.env	최초 커밋	2 hours ago
.gitignore	최초 커밋	2 hours ago
README.md	최초 커밋	2 hours ago

About

No description, website, or topics provided.

Readme Activity 0 stars 1 watching 0 forks

Releases

No releases published [Create a new release](#)

<https://github.com/otzslayer/advanced-mlops/>

본 저장소 Fork

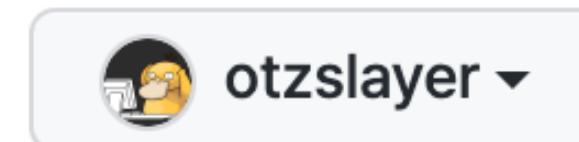
### 1.3 개발환경 설정

## 저장소 생성

### Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project. [View existing forks.](#)

**Owner \***



**Repository name \***

/ advanced-mlops

By default, forks are named **advanced-mlops**. Your new repository will be created as **advanced-mlops**. You can change the name to distinguish it further.

**Description (optional)**

**Copy the main branch only**

Contribute back to GoogleCloudPlatform/mlops-on-gcp by adding your own branch. [Learn more.](#)

You are creating a fork in your personal account.

**Create fork**

## 1.3 개발환경 설정

# GitHub Codespaces



## GitHub Codespaces 사용 시 주의사항

- GitHub Codespaces는 아무 것도 하지 않은 채 30분이 지나면 세션이 종료됩니다.
- GitHub Codespaces를 종료하고 다시 켜실 때 반드시 <https://github.com/codespaces>에서 기존 Codespace를 다시 실행하셔야 합니다.
- 모든 실습 코드는 <https://github.com/otzslayer/lgcns-advanced-mlops>에 있습니다!
- 모든 실습은 로컬에서 진행합니다!



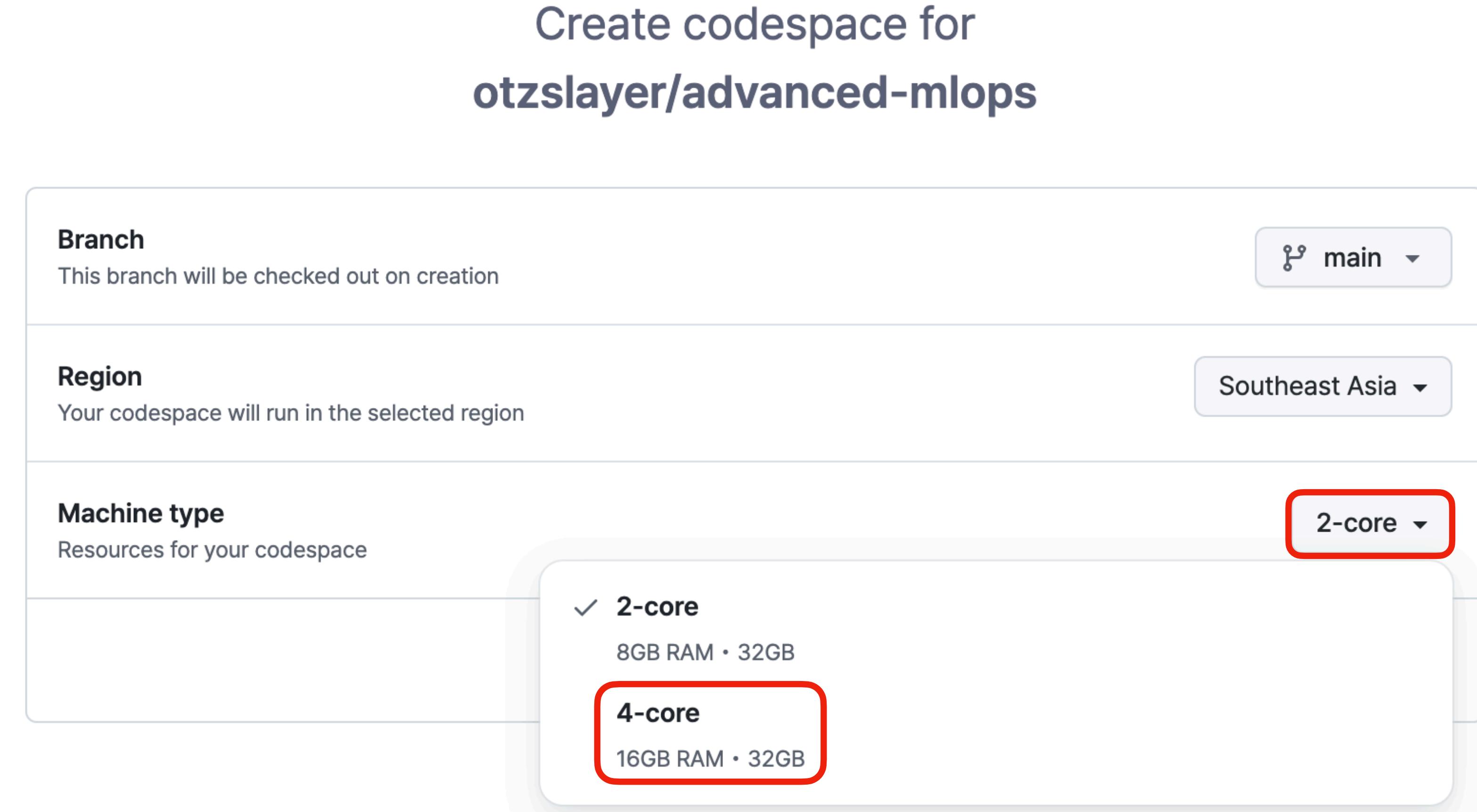
## 1.3 개발환경 설정

# Codespaces 생성

The screenshot shows a GitHub repository page for 'otzslayer / advanced-mlops'. The 'Code' tab is selected. In the top right, there are buttons for 'Pin', 'Unwatch 1', and 'Fork 0'. Below the navigation bar, it shows 'main' branch, '1 Branch', and '0 Tags'. On the left, there's a list of files: 'api', 'db', 'pipelines', 'utils', '.env', '.gitignore', and 'README.md', all marked as '최초 커밋' (first commit). On the right, under the 'About' section, there's a 'Codespaces' dropdown menu. The 'Codespaces' tab is selected, showing a list of existing workspaces: 'otzslayer 최초 커밋' (with sub-folders for api, db, pipelines, utils, .env, .gitignore, and README.md). Below this, it says 'No codespaces' and has a 'Create codespace on main' button. A context menu is open over the 'Create codespace on main' button, with the 'New with options...' item highlighted by a red box. Other options in the menu include 'Configure dev container', 'Set up prebuilds', 'Manage codespaces', 'Share a deep link', and 'What are codespaces?'. The bottom right corner shows a 'Packages' section with 'No packages published'.

## 1.3 개발환경 설정

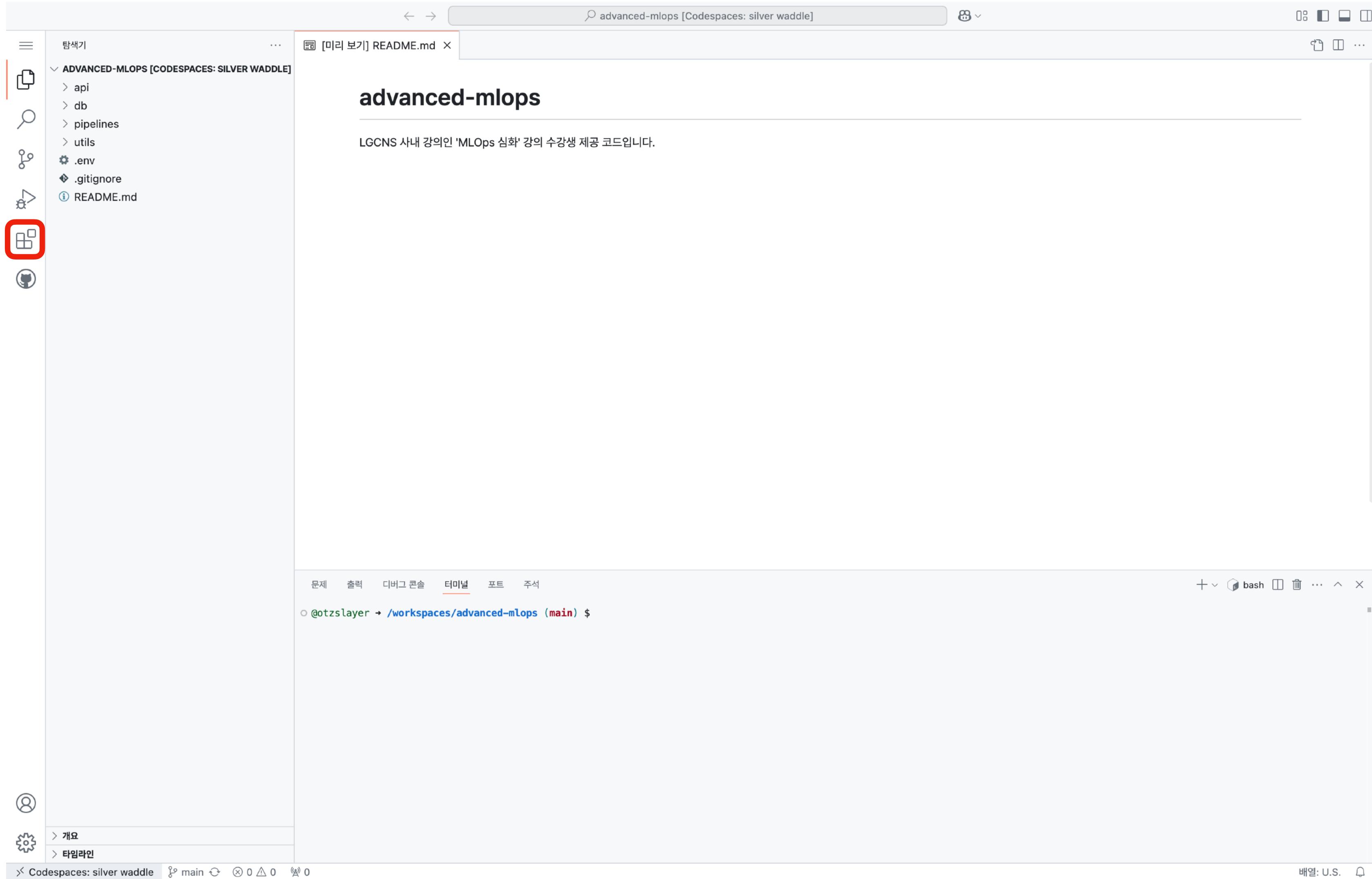
# Codespaces 생성



**4-core / 16GB RAM / 32GB Disk 설정  
(무료로 30시간 사용 가능)**

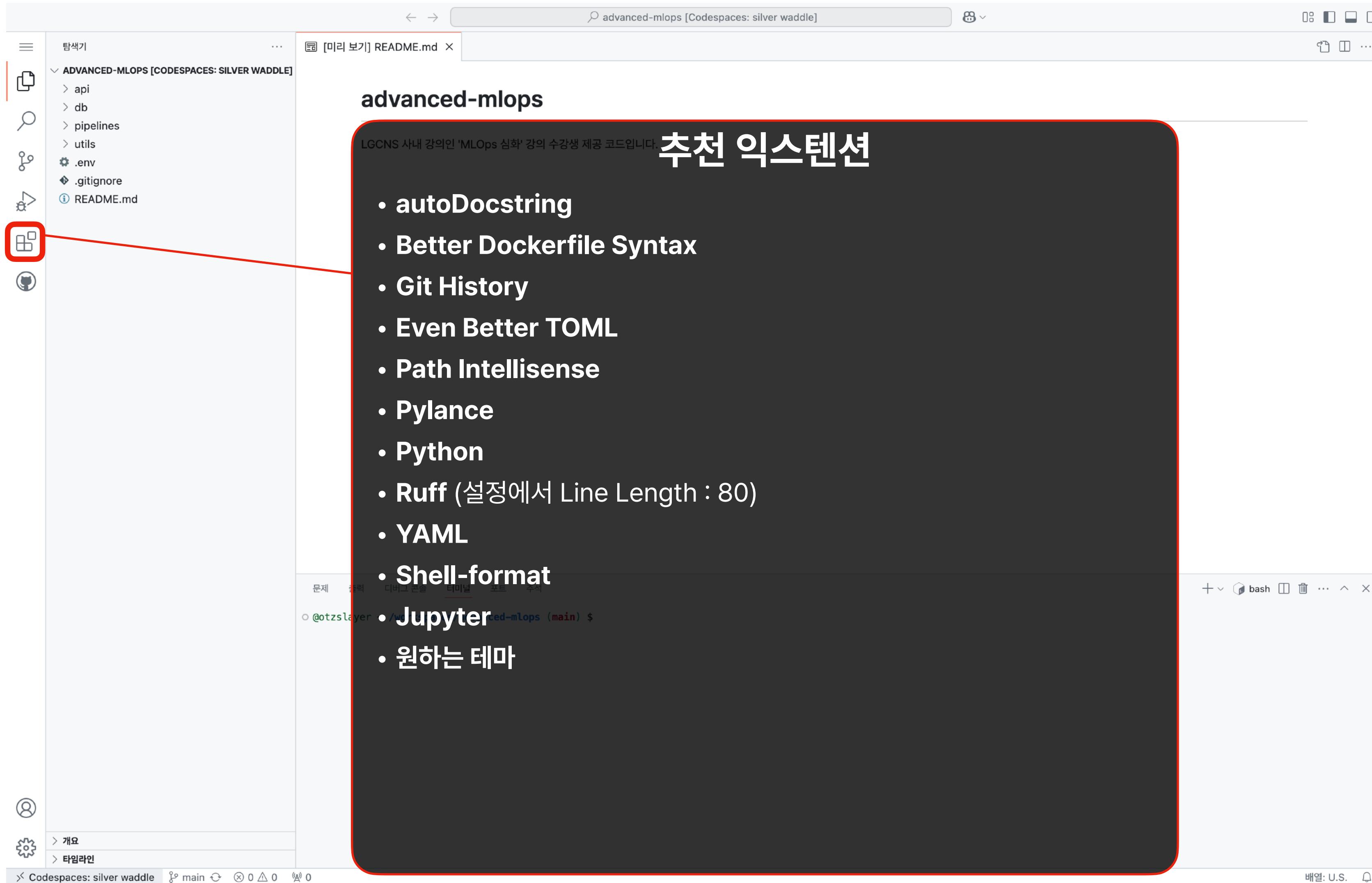
### 1.3 개발환경 설정

## Codespaces 생성



### 1.3 개발환경 설정

## Codespaces 생성



### 1.3 개발환경 설정

## 가상환경 설정

---

```
# GitHub Codespace 내 Docker 버전은 현재 Docker compose 관련 버그가 있어 수정이 필요
# 아래 명령어를 실행한 후 ~/.bashrc에 추가
$ export COMPOSE_BAKE=false

# Ubuntu apt 저장소 업데이트
$ sudo apt-get update

# 일부 라이브러리 설치
$ sudo apt-get install -y make build-essential libssl-dev libreadline-dev wget curl libmysqlclient-dev
$ sudo apt-get upgrade gcc g++

# Python 라이브러리 설치 전 환경 변수 설정
$ export CFLAGS=""
$ export CXXFLAGS=""

# 서버 시간을 KST로 변경
$ sudo rm /etc/localtime
$ sudo ln -s /usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

### 1.3 개발환경 설정

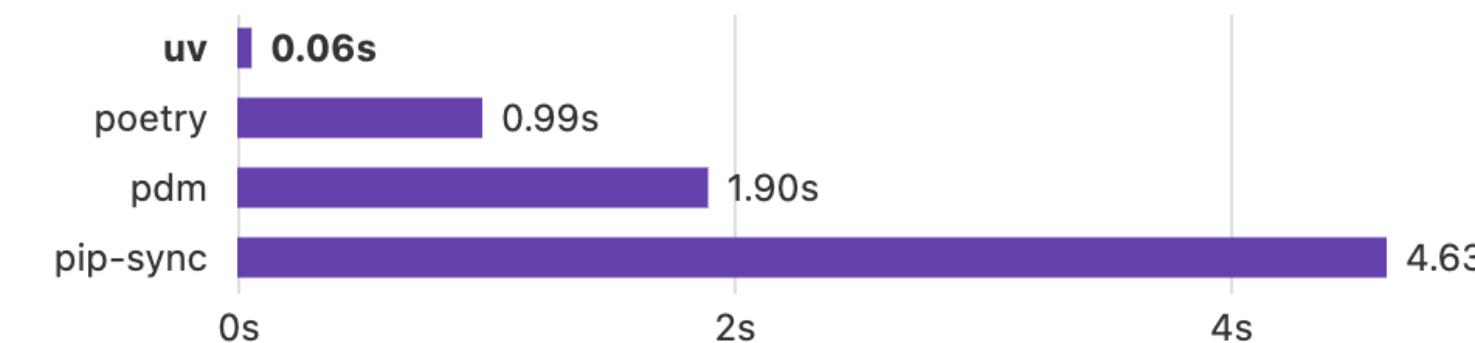
## 가상환경 설정

---

### uv



An extremely fast Python package and project manager, written in Rust.



*Installing [Trio](#)'s dependencies with a warm cache.*

### Highlights

- 🚀 A single tool to replace `pip`, `pip-tools`, `pipx`, `poetry`, `pyenv`, `twine`, `virtualenv`, and more.
- ⚡ [10-100x faster](#) than `pip`.
- 📁 Provides [comprehensive project management](#), with a [universal lockfile](#).
- ⚒️ [Runs scripts](#), with support for [inline dependency metadata](#).
- 🐍 [Installs and manages](#) Python versions.
- 🔧 [Runs and installs](#) tools published as Python packages.
- 🛠️ Includes a [pip-compatible interface](#) for a performance boost with a familiar CLI.
- 🏗️ Supports Cargo-style [workspaces](#) for scalable projects.
- 💾 Disk-space efficient, with a [global cache](#) for dependency deduplication.
- ⬇️ Installable without Rust or Python via `curl` or `pip`.
- 🖥️ Supports macOS, Linux, and Windows.

uv is backed by [Astral](#), the creators of [Ruff](#).

### 1.3 개발환경 설정

## 가상환경 설정

---

```
# uv 설치
$ curl -LsSf https://astral.sh/uv/install.sh | sh
```

```
# 프로젝트 환경 초기화 (Python 3.12.7 설치 포함)
$ uv init --python 3.12.7
$ uv venv --python 3.12.7
$ export PYTHONPATH=/workspaces/advanced-mlops
```

```
# Python이 올바르게 설치되었는지 확인하기 위해 가상환경으로 접근
$ source .venv/bin/activate
$ python
```

```
# Python 실행 시 다음 에러의 발생 여부 확인
# Cannot read termcap database;
# using dumb terminal settings.
# 발생한 경우 code ~/.bashrc 실행하여 해당 파일 맨 마지막에 다음 줄을 추가
export TERMINFO_DIRS=/etc/terminfo:/lib/terminfo:/usr/share/terminfo

# 이후 터미널에서 다음 명령어 실행
$ source ~/.bashrc
```

```
# 불필요한 파일이 있다면 삭제 (ex. main.py)
$ rm main.py
```

### 1.3 개발환경 설정

## 가상환경 설정

---

```
# pyproject.toml 내 의존성 관리를 위해 Airflow 우선 설치
$ uv add apache-airflow==3.0.4

# Airflow 버전 및 Python 버전 설정
$ export AIRFLOW_VERSION=3.0.4
$ export PYTHON_VERSION="$(python -c 'import sys; print(f'{sys.version_info.major}.{sys.version_info.minor}')')"

# Constraints 파일 URL 생성
$ export CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW_VERSION}/constraints-${PYTHON_VERSION}.txt"

# uv를 이용한 Airflow 설치
$ uv pip install "apache-airflow==${AIRFLOW_VERSION}" --constraint "${CONSTRAINT_URL}"
```

### 1.3 개발환경 설정

## 가상환경 설정

---

```
# 사용할 라이브러리 설치
$ uv add apache-airflow-providers-mysql=6.3.3 mysqlclient=2.2.7 numpy=2.3.2 pandas=2.3.1 pymysql=1.1.1 scikit-learn=1.7.1 python-dotenv=1.1.1 mlflow=3.2.0 bentoml=1.4.19 catboost=1.2.8 pip=25.2 tqdm=4.67.1 aiomysql=0.2.0 locust=2.39.0

# 개발용 라이브러리(Jupyter 등) 설치
$ uv add --group jupyter ipykernel ipython
```

### 1.3 개발환경 설정

## 가상환경 설정

```
# 프로젝트 폴더 내 pyproject.toml 파일을 확인해보면 아래와 같이 수정되어 있음

[project]
name = "advanced-mlops"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = "≥3.12.7"
dependencies = [
    "aiomysql=0.2.0",
    "apache-airflow=3.0.4",
    "apache-airflow-providers-mysql=6.3.3",
    "bentoml=1.4.19",
    "catboost=1.2.8",
    "fastapi=0.116.1",
    "locust=2.39.0",
    "mlflow=3.2.0",
    "mysqlclient=2.2.7",
    "numpy=2.3.2",
    "pandas=2.3.1",
    "pip=25.2",
    "pymysql=1.1.1",
    "python-dotenv=1.1.1",
    "scikit-learn=1.7.1",
    "tqdm=4.67.1",
]

[dependency-groups]
jupyter = [
    "ipykernel≥6.29.5",
    "ipython≥8.31.0",
    "jupyter≥1.1.1",
]
```

### 1.3 개발환경 설정

## 가상환경 설정

---

```
# 변경 사항 커밋 후 푸시  
$ git add .  
$ git commit -m "프로젝트 초기화"  
$ git push origin main
```



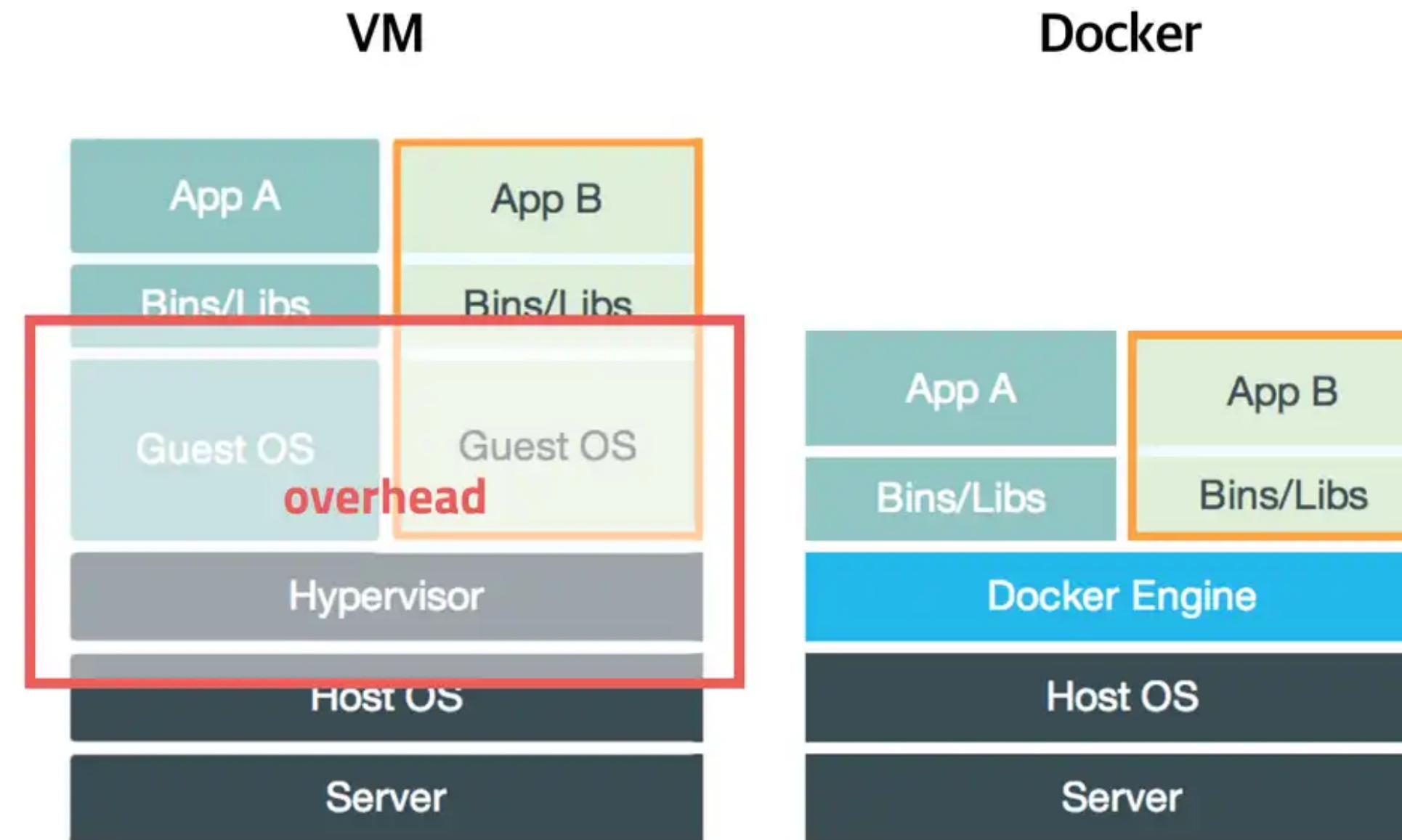
## M2. 활용 도구 익히기

1. Docker
2. SQLAlchemy
3. Airflow

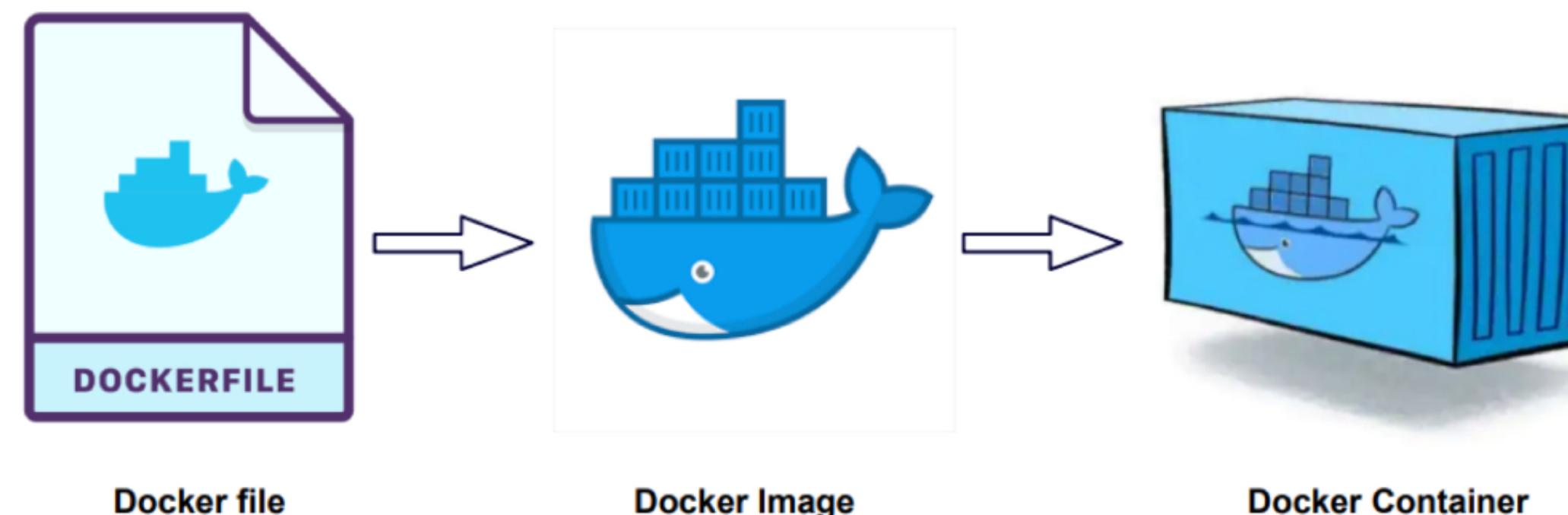
## **2.1 Docker**

## 2.1 Docker

# Docker



- **Docker는 컨테이너 기반의 가상화 플랫폼**
- 기존 가상화는 OS 가상화를 수행함
  - 게스트 OS 전체를 가상화하기 때문에 높은 오버헤드가 발생하고 성능도 높지 않음
- 성능 문제를 해결하기 위해 프로세스를 격리하는 방식 등장
  - **컨테이너 내에 프로세스를 격리**
    - 여러 개의 컨테이너를 띄워도 **성능 손실이 거의 없음**
    - 이런 방법을 잘 정리하여 다양한 기능을 추가한 것이 바로 **Docker**



- **Docker의 작동 방식은 매우 간단함**
  - Dockerfile에 Docker 이미지에 대한 명세를 자세하게 적기만 하면 됨
  - 이후 몇 개의 간단한 명령어로 이미지 생성 후 컨테이너를 실행

## 2.1 Docker

# Docker를 사용하는 이유

1

## 📌 OS 독립성

- Docker 컨테이너는 애플리케이션과 모든 종속성을 함께 패키징하여 **어떤 OS에서도** 동일한 환경을 제공할 수 있음
- 개발 환경과 운영 환경이 다르더라도, 또는 모든 개발자가 다른 OS에서 개발하더라도 종속성 문제 없이 동일한 결과를 보장할 수 있음

2

## 📌 빠른 개발/배포 가능

- Docker를 이용해 컨테이너를 빠르게 생성, 실행, 제거할 수 있기 때문에, ML 프로젝트의 **반복적인 작업이 단순화되고 가속화됨**
- GitHub Actions, Jenkins와 같은 CI/CD 파이프라인과 결합하면 매우 간편하고 빠른 배포가 가능함

3

## 📌 높은 확장성과 유연성

- 사용자가 많은 환경을 위해 개발/배포하는 경우 **Kubernetes와 같은 도구와 결합해 쉽게 확장하고 유연하게 배포**할 수 있음
- 특히 운영 환경에서 사용량에 맞춰 컨테이너 수를 자동으로 조절하고, 부하에 따라 로드 밸런싱 구현도 용이함



## 2.1 Docker

# Dockerfile

# Dockerfile

- 원하는 Docker 이미지를 제작하기 위한 설정 파일
  - 이 파일만 있으면 Docker 이미지를 바로 빌드할 수 있음
  - (명령어 인자) 형태의 명령문으로만 구성되어 있음

```

Dockerfile

1 FROM python:3.12
2 WORKDIR /usr/local/app
3
4 # Install the application dependencies
5 COPY requirements.txt .
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 # Copy in the source code
9 COPY src ./src
10 EXPOSE 5000
11
12 # Setup an app user so the container doesn't run as the root user
13 RUN useradd app
14 USER app
15
16 CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]

```

<b>FROM</b>	• 생성할 이미지의 <b>기본 이미지를 불러오는 명령어</b>
<b>WORKDIR</b>	• 컨테이너 내부에서의 <b>작업 디렉토리를 전환하는 명령어</b>
<b>COPY/ADD</b>	• 로컬 호스트의 파일이나 디렉터리를 컨테이너 내 파일 시스템으로 <b>복사하거나 추가하기 위해 사용</b>
<b>RUN</b>	• 쉘(Shell)에서 명령을 입력하기 위한 명령어
<b>EXPOSE</b>	• 컨테이너 <b>내부에서 포트를 열어주는 명령어</b> • 컨테이너 외부에서 접근하려면 <b>명령어에 옵션을 추가해야 함</b>
<b>USER</b>	• 컨테이너에서 사용할 기본 사용자명
<b>CMD</b>	• 컨테이너를 실행할 때의 디폴트 커맨드나 파라미터 • 컨테이너 실행 시 주어지는 인자에 덮어씌워질 수 있음
<b>ENV</b>	• 컨테이너 내 환경 변수를 설정하기 위한 명령어 • ENV <key>=<value> 형태로 사용
<b>ARG</b>	• Docker 이미지를 빌드할 때 넘길 인자를 정의하는 명령어 • ARG <name>=<default value> 형태로 사용

## 2.1 Docker

# Docker 기본 명령어

---

명령어	설명
<b>docker run</b>	컨테이너를 생성하고 실행합니다. ( <code>docker run [OPTIONS] IMAGE [COMMAND]</code> )
<b>docker ps</b>	현재 실행 중인 컨테이너 목록을 확인합니다.
<b>docker ps -a</b>	종료된 컨테이너를 포함하여 모든 컨테이너 목록을 확인합니다.
<b>docker images</b>	로컬에 저장된 Docker 이미지 목록을 확인합니다.
<b>docker rm</b>	특정 컨테이너를 삭제합니다. ( <code>docker rm [CONTAINER_ID]</code> )
<b>docker rmi</b>	특정 Docker 이미지를 삭제합니다. ( <code>docker rmi [IMAGE_ID]</code> )
<b>docker exec</b>	실행 중인 컨테이너 내부에서 명령을 실행합니다. ( <code>docker exec -it [CONTAINER_ID] /bin/sh</code> )

## 2.1 Docker

# Docker 기본 명령어



The screenshot shows a macOS terminal window with a dark theme. The title bar indicates the current directory is the user's home folder (~). The terminal output is as follows:

```
> docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (arm64v8)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

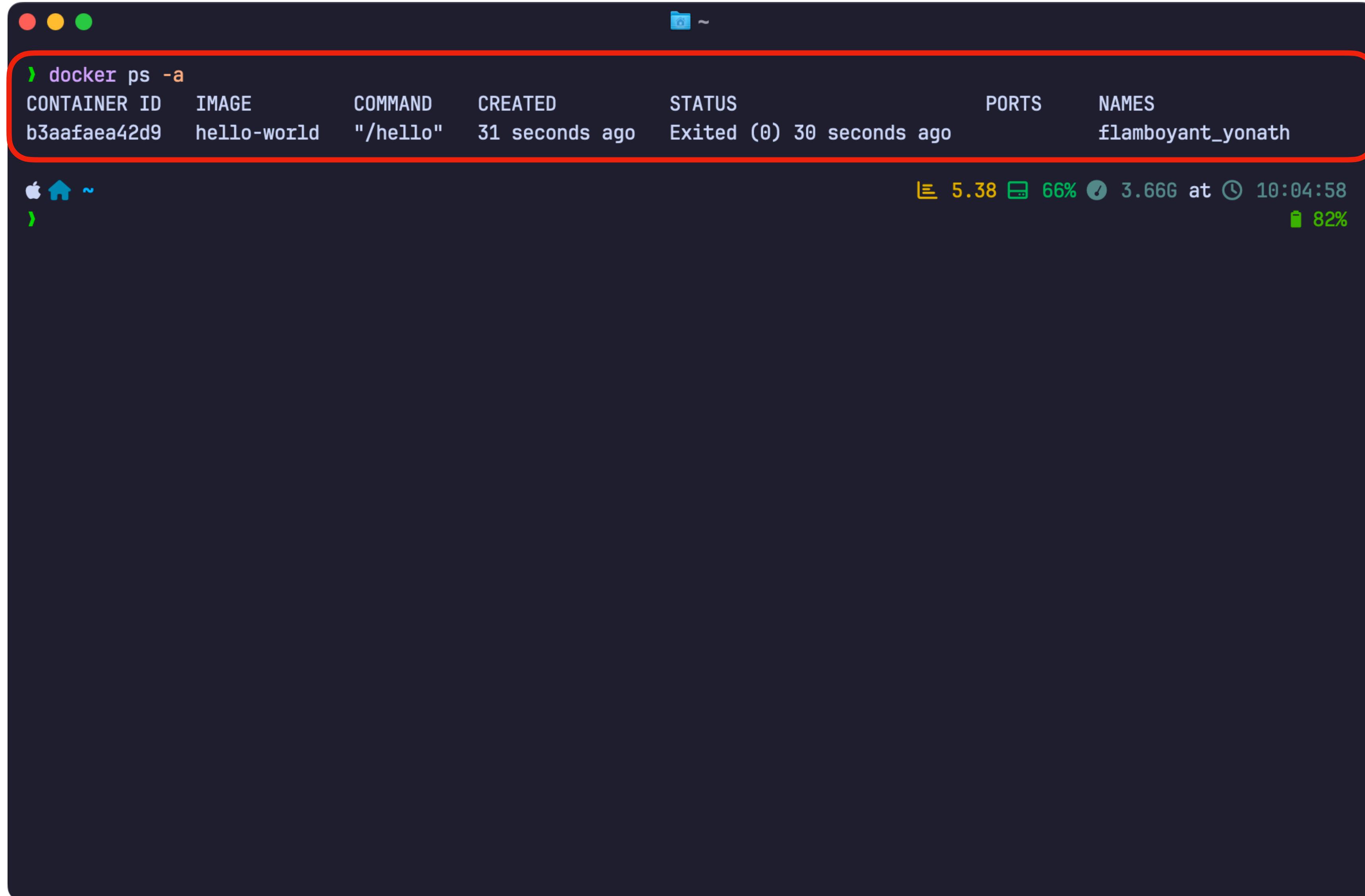
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

The bottom of the screen shows the macOS system status bar with battery level (3.83), battery percentage (66%), signal strength (3.96G), time (at 16:24:15), and battery percentage again (39%).

## 2.1 Docker

# Docker 기본 명령어



```
› docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b3aaafaea42d9 hello-world "/hello" 31 seconds ago Exited (0) 30 seconds ago
flamboyant_yonath
```

## 2.1 Docker

# Docker 기본 명령어

The screenshot shows a macOS terminal window with a dark theme. The terminal output is as follows:

```
> docker ps -a
CONTAINER ID        IMAGE       COMMAND      CREATED     STATUS          PORTS     NAMES
b3aaafaea42d9    hello-world "hello"    17 minutes ago   Exited (0) 7 minutes ago
flamboyant_yonath

> docker rm b3
b3

> docker ps -a
CONTAINER ID        IMAGE       COMMAND      CREATED     STATUS          PORTS     NAMES
```

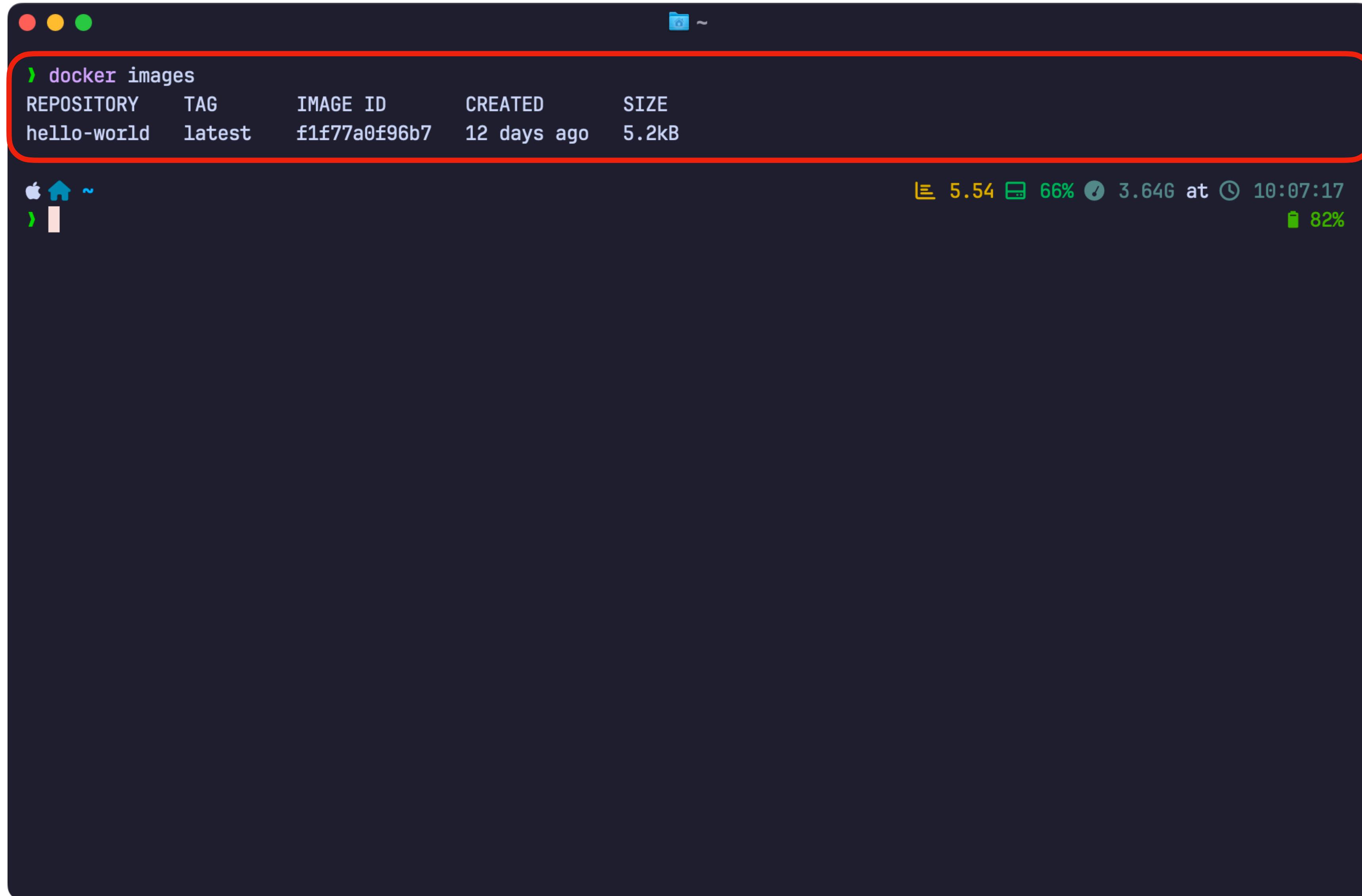
The first command, `docker ps -a`, lists all containers. The container with ID `b3aaafaea42d9` is highlighted with a red box. The second command, `docker rm b3`, removes this container. The third command, `docker ps -a`, shows that the container has been removed.

At the bottom of the terminal window, there is a system status bar displaying:

- Apple icon
- Home icon
- ~
- Battery icon: 4.61 66% 3.56G at 10:22:24
- Battery icon: 82%

## 2.1 Docker

# Docker 기본 명령어



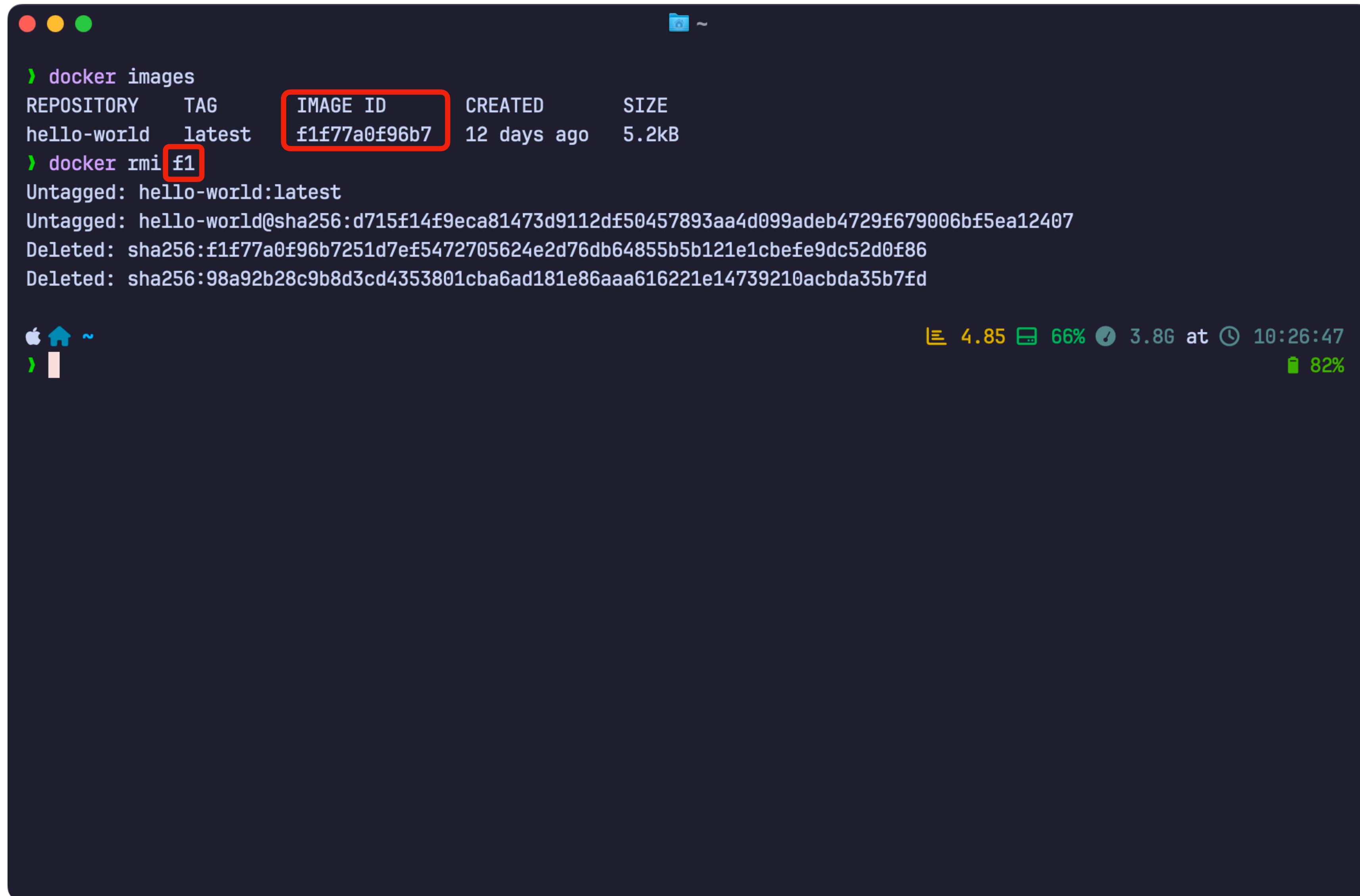
```
› docker images
REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
hello-world     latest       f1f77a0f96b7  12 days ago   5.2kB
```

apple ~

5.54 66% 3.64G at 10:07:17  
82%

## 2.1 Docker

# Docker 기본 명령어



The screenshot shows a macOS terminal window with a dark theme. The user has run the command `docker images`, which lists a single image:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	f1f77a0f96b7	12 days ago	5.2kB

Then, the user runs `docker rmi f1`. The `IMAGE ID` column is highlighted with a red box. The output shows the image is untagged and then deleted, along with its corresponding SHA-256 digest.

At the bottom right of the terminal window, there is a system status bar displaying battery level (82%), signal strength, and system time (10:26:47).

```
> docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
hello-world      latest    f1f77a0f96b7  12 days ago  5.2kB
> docker rmi f1
Untagged: hello-world:latest
Untagged: hello-world@sha256:d715f14f9eca81473d9112df50457893aa4d099adef4729f679006bf5ea12407
Deleted: sha256:f1f77a0f96b7251d7ef5472705624e2d76db64855b5b121e1cbefef9dc52d0f86
Deleted: sha256:98a92b28c9b8d3cd4353801cba6ad181e86aaa616221e14739210acbda35b7fd
```

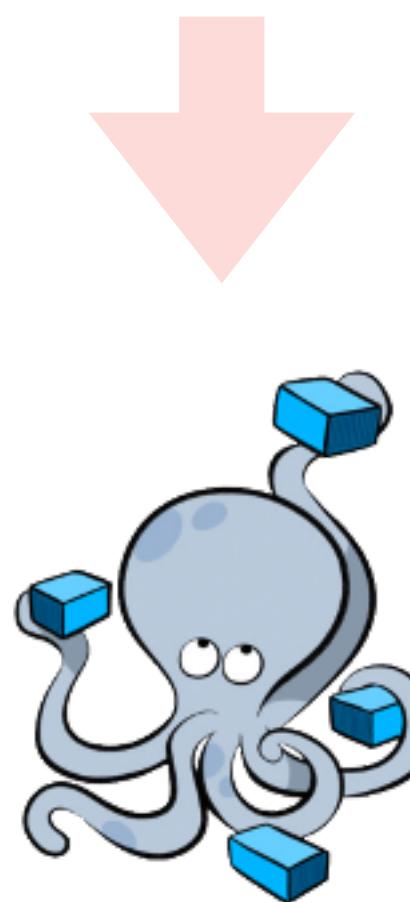
## 2.1 Docker

# Docker Compose

만약 한 번에 여러 개의 컨테이너를 띄워야 한다면?

번거롭게 컨테이너 실행 명령어를 여러 번 수행하는 것보다

**Docker Compose**를 활용하여 한 번에 띄우는 것이 편함!



**docker-compose.yml** 파일을 적절한 위치에 두고

몇 가지 명세를 작성한 다음

**docker compose up** 명령어 실행하면 끝!

### db/docker-compose.yml

```

1 services:
2   db:
3     image: mariadb:10.8.2-rc-focal
4     container_name: mariadb
5     hostname: mariadb
6     volumes:
7       - ${PWD}/data:/tmp
8       - ${PWD}/docker-entrypoint-initdb.d:/docker-entrypoint-initdb.d
9     restart: always
10    ports:
11      - "3306:3306"
12    environment:
13      MARIADB_ROOT_PASSWORD: root
14    networks:
15      mlops_network:
16    adminer:
17      image: adminer
18      container_name: mariadb_admin
19      hostname: adminer
20      restart: always
21      ports:
22        - "8089:8080"
23    networks:
24      mlops_network:
25    networks:
26      mlops_network:
27        name: mlops_network
28        external: true

```

## 2.1 Docker

# Docker Compose

컨테이너로 실행할 서비스 목록을 아래에 정의

서비스 이름

사용할 이미지 이름 (기본 이미지 이름:버전 형식으로 작성)

컨테이너 이름

볼륨 마운트 설정

A:B 형태로 작성하고, 로컬 디렉토리인 A가 컨테이너 내부의 B 디렉토리로 마운트 된다는 의미

컨테이너 재시작 정책

포트 매팅 설정

A:B 형태로 작성하고, 외부에서는 A 포트로 접근 가능

환경 변수 설정

네트워크 설정

동일한 네트워크에 있는 다른 컨테이너에서 해당 컨테이너 접근 가능

네트워크 설정

네트워크 이름

이미 존재하는 네트워크를 사용하도록 지정

서비스 실행 전에 docker network create <network\_name> 실행 필요

db/docker-compose.yml

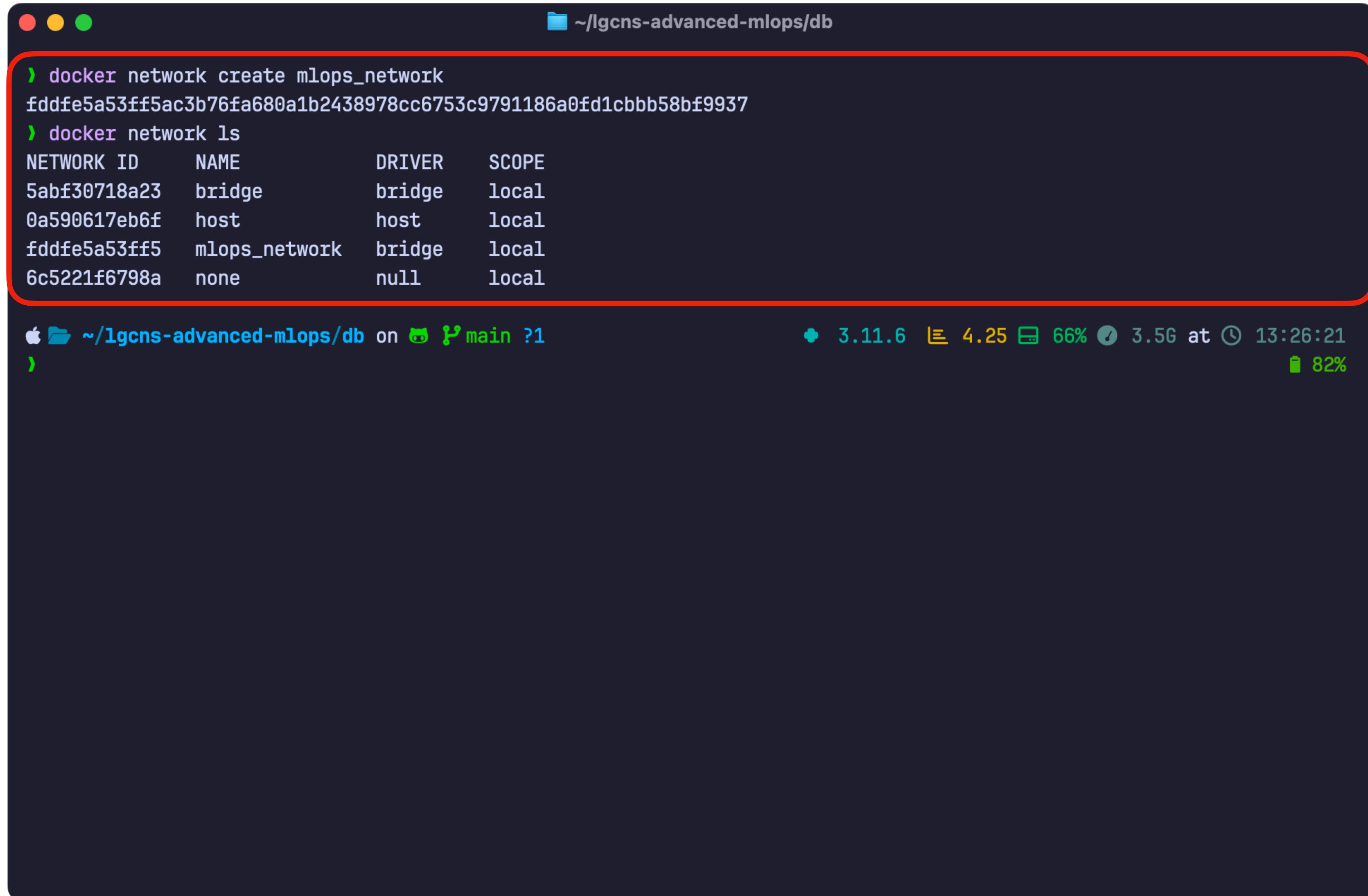
```

1 services:
2   db:
3     image: mariadb:10.8.2-rc-focal
4     container_name: mariadb
5     hostname: mariadb
6     volumes:
7       - ${PWD}/data:/tmp
8       - ${PWD}/docker-entrypoint-initdb.d:/docker-entrypoint-initdb.d
9     restart: always
10    ports:
11      - "3306:3306"
12    environment:
13      MARIADB_ROOT_PASSWORD: root
14    networks:
15      mllops_network:
16        adminer:
17          image: adminer
18          container_name: mariadb_adminer
19          hostname: adminer
20          restart: always
21          ports:
22            - "8089:8080"
23        networks:
24          mllops_network:
25        networks:
26          mllops_network:
27            name: mllops_network
28            external: true

```

## 2.1 Docker

# Docker Compose



The screenshot shows a macOS terminal window with a dark theme. The title bar indicates the path: `~/lgcns-advanced-mlops/db`. The main pane contains the following command-line session:

```
> docker network create mlops_network
fddfe5a53ff5ac3b76fa680a1b2438978cc6753c9791186a0fd1cbbb58bf9937
> docker network ls
NETWORK ID      NAME      DRIVER      SCOPE
5abf30718a23    bridge    bridge      local
0a590617eb6f    host      host       local
fddfe5a53ff5    mlops_network    bridge      local
6c5221f6798a    none      null       local
```

The output of the `docker network ls` command is highlighted with a red rounded rectangle.

At the bottom of the terminal window, there is a status bar with the following information:

- Apple icon
- File icon (~/lgcns-advanced-mlops/db)
- Git icon (main)
- Terminal icon (main ?1)
- CPU icon (3.11.6)
- Memory icon (4.25)
- Battery icon (66%)
- Network icon (3.5G at 13:26:21)
- Battery level (82%)

## 2.1 Docker

# Docker Compose로 mariadb와 Adminer 컨테이너 띄우기

---

```
# db 폴더로 접근하여 컨테이너를 띄우면 됨
$ cd db

# 만약 네트워크를 생성하지 않았다면 생성 필요
$ docker network create mlops_network

# -d 옵션을 추가하여 백그라운드에서 실행되도록 함
$ docker compose up -d
```

## 2.1 Docker

# Docker Compose로 mariadb와 Adminer 컨테이너 띄우기

```
❯ docker compose up -d
[+] Running 20/20
  ✓ db Pulled                                27.3s
    ✓ d4ba87bb7858 Pull complete               10.8s
    ✓ 932d0da43793 Pull complete               10.8s
    ✓ dc35c765a732 Pull complete               10.9s
    ✓ ed88bae007bd Pull complete               11.0s
    ✓ 7734d2f7be98 Pull complete               11.0s
    ✓ e24600bac48c Pull complete               11.8s
    ✓ da05700210f6 Pull complete               11.8s
    ✓ ce763ae3d194 Pull complete               11.8s
    ✓ 1d496894956c Pull complete               24.6s
    ✓ c5ff3d92cd0a Pull complete               24.6s
    ✓ b65305c7f16f Pull complete               24.6s
  ✓ adminer Pulled                            15.6s
    ✓ 2c9750102c61 Pull complete               11.7s
    ✓ 1eb6bc35ca96 Pull complete               12.9s
    ✓ baf5a77f4b47 Pull complete               12.9s
    ✓ de9e0e1297b2 Pull complete               12.9s
    ✓ bd2fd75f19c8 Pull complete               12.9s
    ✓ 8fd82873adc1 Pull complete               13.0s
    ✓ 61f74b8ed8bd Pull complete               13.0s
[+] Running 2/2
  ✓ Container mariadb           Started      0.6s
  ✓ Container mariadb_adminer  Started      0.6s

❯
took ✎ 28s • 3.11.6 | 5.48 □ 66% ⚡ 3.13G at ⏲ 13:53:30
  82%
```

## 2.1 Docker

# Docker Compose로 mariadb와 Adminer 컨테이너 띄우기

The screenshot shows a GitHub Codespace interface. The left sidebar displays a project structure for 'ADVANCED-MLOPS [CODESPACES: SILVER WADDLE]'. The 'docker-compose.yml' file is selected in the tree view. The main area shows the content of the 'db' service's docker-compose.yml file:

```
1 services:
2   db:
3     image: mariadb:10.8.2-rc-focal
4     container_name: mariadb
5     hostname: mariadb
6     volumes:
7       - ${PWD}/data:/tmp
8       - ${PWD}/docker-entrypoint-initdb.d:/docker-entrypoint-initdb.d
9     restart: always
10    ports:
11      - "3306:3306"
12    environment:
13      MARIADB_ROOT_PASSWORD: root
14    networks:
15      - mlops_network
16  adminer:
17    image: adminer
18    container_name: mariadb_admin
19    hostname: adminer
20    restart: always
21    ports:
22      - "8089:8080"
23    networks:
24      - mlops_network
25  networks:
26    - mlops_network
27    name: mlops_network
28    external: true
29
```

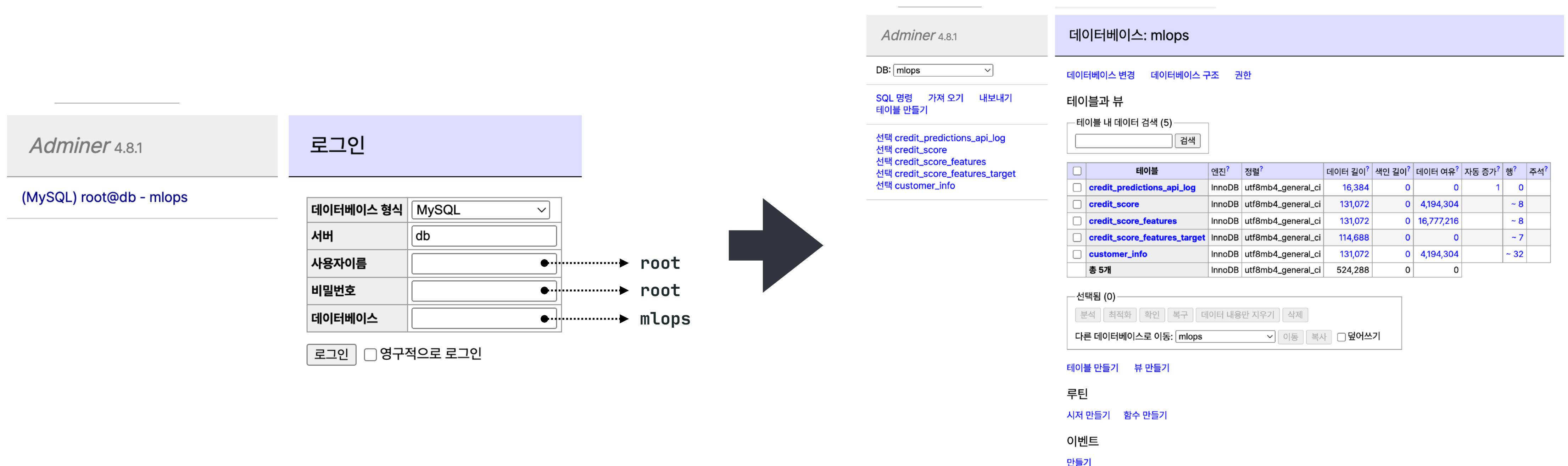
Below the code editor, a port mapping table is displayed. The '포트' (Port) tab is selected, showing two entries:

포트	전달된 주소	실행 중인 프로세스	표시 여부	원본
3306	https://silver-waddle-7vx... ...	/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-po... ...	Private	자동 전달됨
8089	https://silver-waddle-7vx... ...	/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-po... ...	Private	자동 전달됨

The rows for ports 3306 and 8089 are highlighted with a red box.

## 2.1 Docker

# Docker Compose로 mariadb와 Adminer 컨테이너 띄우기



## 2.1 Docker

# Docker Compose로 mariadb와 Adminer 컨테이너 띄우기

**Adminer 4.8.1**

DB: mlops

SQL 명령 가져 오기 내보내기  
테이블 만들기

선택 credit\_predictions\_api\_log  
선택 credit\_score  
선택 credit\_score\_features  
선택 credit\_score\_features\_target  
선택 customer\_info

**SQL 명령**

```
SELECT * FROM mlops.customer_info LIMIT 10
```

id	customer_id	date	name	ssn
100127	23186	2025-06-12	Gillams	486471751
100138	6601	2025-05-12	Cableo	534541028
100188	42573	2025-07-30	Nick Edwardsl	190660409
100234	36210	2025-05-30	Hideyukih	200137091
100236	36210	2025-07-30	Hideyukih	200137091
100369	26053	2025-08-16	Clarer	78370144
10041	32609	2025-04-12	Saphirq	707441618
100417	16265	2025-08-12	Luciag	895180686
100499	29935	2025-06-16	LaCapraz	437790896
100627	6356	2025-02-16	Michele Kambasy	532177094

10개 행 (0.002 초) 편집, Explain, 내보내기

```
SELECT * FROM mlops.customer_info LIMIT 10
```

실행
행 제약: 
 오류의 경우 중지
 오류 만 표시

이력

## **2.2 SQLAlchemy**

## 2.2 SQLAlchemy

# SQLAlchemy?

## ORM (Object Relational Mapping)

- 애플리케이션과 데이터베이스를 연결할 때 **SQL 언어가 아닌 애플리케이션의 개발 언어로 데이터베이스를 접근**하도록 도와주는 도구
- 개발 언어의 일관성과 가독성을 높여주는 장점을 갖고 있음

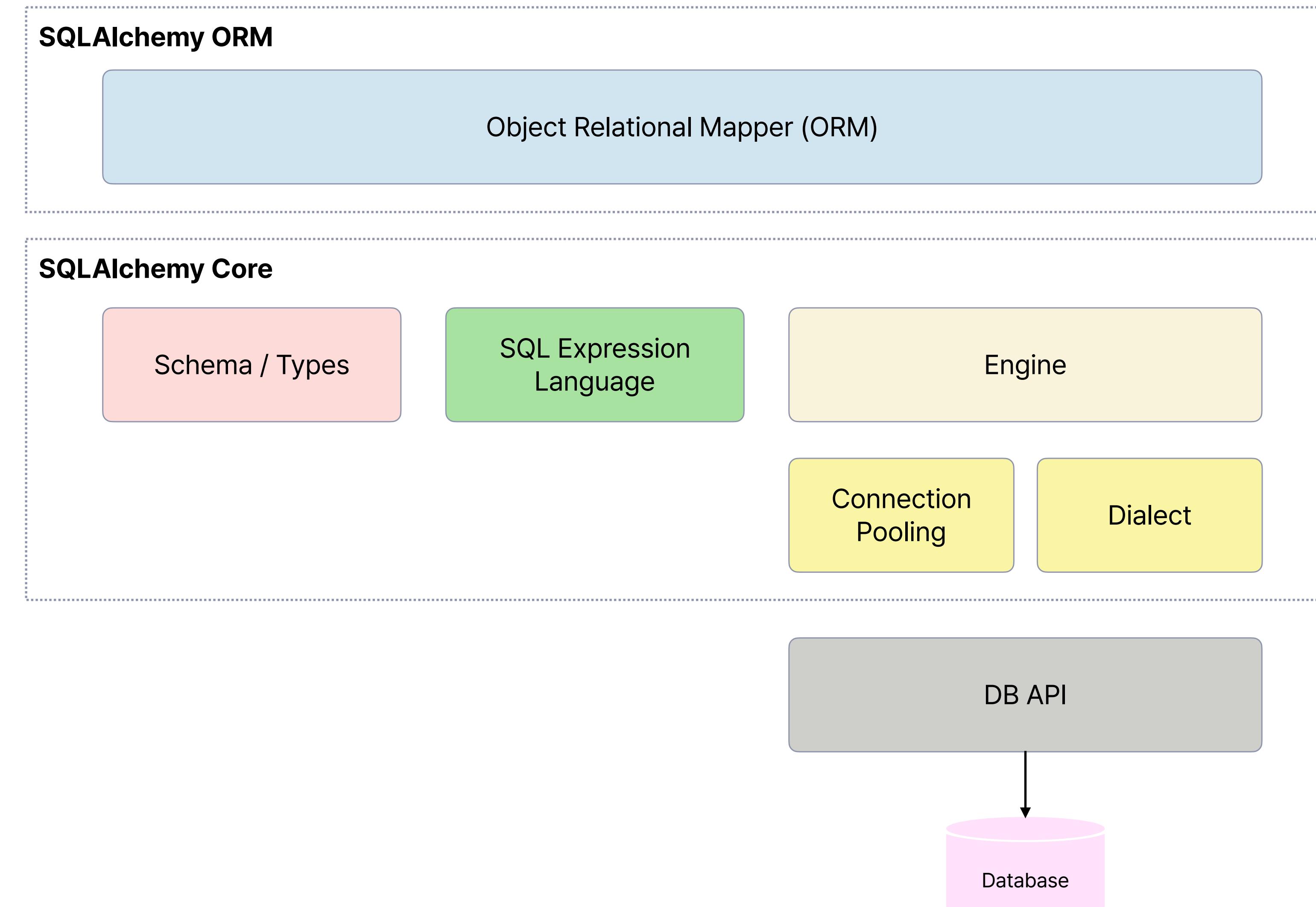
Python에서 가장 널리 사용되는 **ORM**

### 📌 SQLAlchemy 사용 시 주의점

- **Pandas**와의 버전 호환성 주의
    - Pandas 1.X 버전과 SQLAlchemy 1.X.X 버전이 호환됨
    - Pandas 2.X 버전과 SQLAlchemy 2.X.X 버전이 호환됨
    - 이외 경우에는 일부 호환되나 주의가 필요함
  - **의존성 충돌** 주의
    - Airflow의 최신 버전이더라도 SQLAlchemy 2.X.X 버전과 호환이 안됨
    - 따라서 본 강의에서는 SQLAlchemy 1.X.X 버전을 사용
- ➡ 백엔드는 **최대한 최신의 SQLAlchemy를 사용**할 것
- 💡 Airflow 관련 프로젝트와 백엔드를 **다른 가상환경으로** 쓰면 됨

## 2.2 SQLAlchemy

# SQLAlchemy 구성



## 2.2 SQLAlchemy

# SQLAlchemy 사용해서 데이터 불러오기

- SQLAlchemy만을 이용하여 데이터를 불러올 수도 있으나 **Pandas와 통합하여 데이터를 쉽게 불러올 수 있음**
- SQLAlchemy의 모든 동작은 기본적으로 다음 순서를 따름

1. URL 또는 접속 정보를 이용하여 데이터베이스와 연결을 맺는 엔진을 생성

**{dialect}+{driver}://{{username}}:{password}@{{host}}:{{port}}/{{database}}**

2. 엔진에 `connect()` 메서드를 활용해 **Connection** 연결 객체를 통해 데이터베이스와 상호작용

```
import pandas as pd
from sqlalchemy import create_engine, text

# url에 사용하는 dialect와 driver는 각각 mysql과 pymysql
# 컨테이너를 띄울 때 사용한 사용자명과 비밀번호, 포트 번호와 데이터베이스 이름을 추가하여 url 작성
url = "mysql+pymysql://root:root@localhost:3306/mllops"
engine = create_engine(url)

sql = """select *
from mllops.customer_info
"""

# Context Manager (with 절)을 사용하는 것을 권장
# 트랜잭션 단위로 작업을 구분하기 쉽고 더 나아가 복잡한 작업에서 세션 내 연결 관리가 간단해짐
with engine.connect() as conn:
    df = pd.read_sql(sql, con=conn.connection)
```

## 2.2 SQLAlchemy

# 데이터 살펴보기

---

## **2.3 Airflow**

## 2.3 Airflow

# Airflow란

---



- 배치 지향 워크플로우를 개발, 스케줄링, 모니터링하기 위한 오픈소스 플랫폼
- Airflow 내 다양한 Python 프레임워크를 통해 원하는 대로 워크플로우를 구성할 수 있음

## 주요 용어

### DAG (Directed Acyclic Graph)

1

- Airflow의 핵심 개념으로 Task를 종속성과 관계로 정리해 어떻게 실행해야 하는지 정의한 것
- 방향성이 있고 순환이 없는 그래프 형태
- 각 DAG는 Python 코드로 작성하고 다양한 메타데이터로 정의함

2

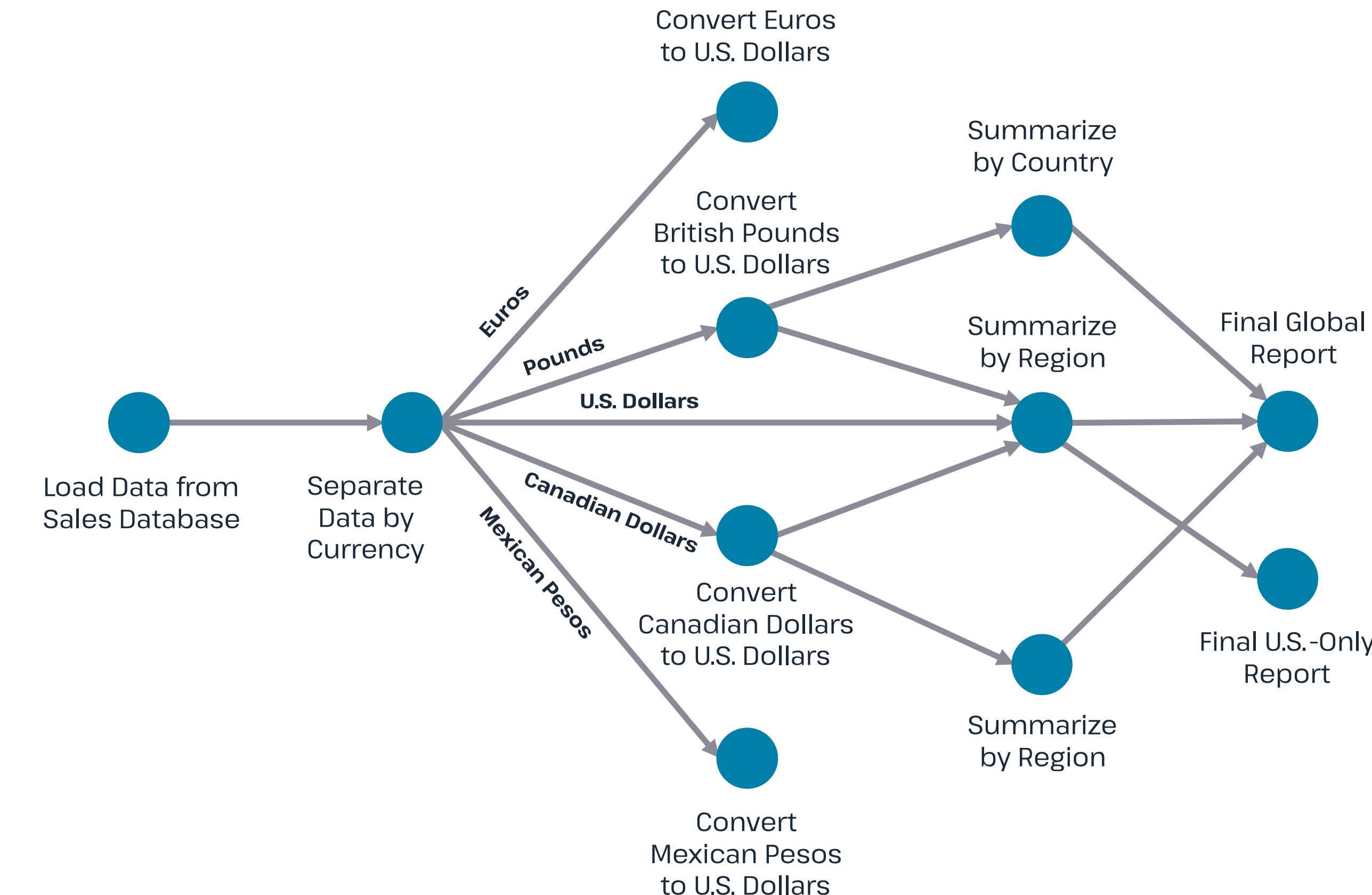
### Task

- DAG는 하나 이상의 Task로 구성되며, DAG에서 실행하는 작업 단위
- 일반적으로 Operator 클래스로 정의되고, DAG 객체에 포함됨

## 2.3 Airflow

# DAG

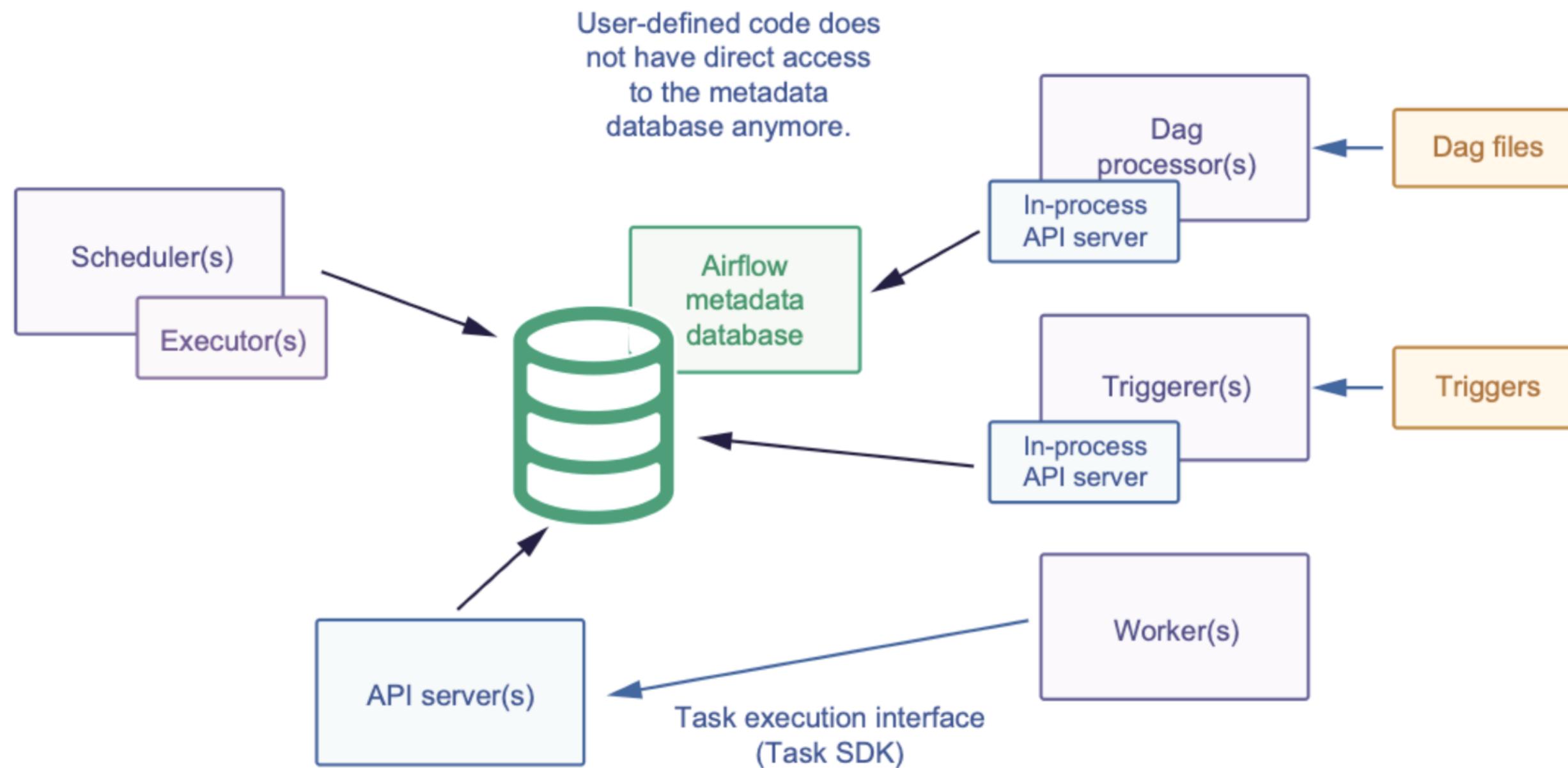
---



## 2.3 Airflow

# Airflow 아키텍처

### Airflow 3 architecture



- DAG Directory**
  - DAG를 보관하는 디렉토리
- Metadata Database**
  - 모든 컴포넌트가 공유하는 메타데이터를 저장하는 데이터베이스
- API Server**
  - UI와 REST API를 제공하는 컴포넌트
  - Airflow 3.X 버전에서 도입
  - 과거 Webserver의 역할을 하며, `airflow api-server` 명령어로 실행
- Scheduler**
  - DAG의 실행을 계획하고, Task를 관리함
  - DAG가 실행될 때마다 실행할 Task를 정의하며, Task는 Executor에 따라 다른 방식으로 실행됨
- DAG Processor**
  - DAG 파일을 파싱해 데이터베이스에 업데이트하는 역할을 하는 필수 컴포넌트
  - Airflow 3.X 버전에서 도입
  - `airflow dag-processor` 명령어로 별도 실행
- Worker**
  - 실제 Task를 처리하는 컴포넌트
  - Executor 종류에 따라 다른 방식으로 작동함
- Executor**
  - Task의 실행 방법을 정의함
  - 다양한 Executor가 있어서 상황에 맞게 설정하여 사용

## 2.3 Airflow

# Airflow 시작하기 – (1) 기본 설정

```
# 기본 설정
# Airflow가 기본적으로 사용하는 폴더 경로 설정
$ mkdir -p ~/airflow
$ export AIRFLOW_HOME=~/airflow

# Codespace 환경에서는 보안 설정을 일부 바꿔줘야 함
$ airflow config list --defaults > "${AIRFLOW_HOME}/airflow.cfg"

# 설정 파일을 열어 일부 설정 변경
$ code ${AIRFLOW_HOME}/airflow.cfg

# 다음 내용을 주석 제거 후 변경
# 1. default_timezone = utc 에서 Asia/Seoul 로 변경
# 2. sqlalchemy_conn = mysql+pymysql://root:root@localhost:3306/airflow 로 변경

# 가상환경 내에서 아래 명령어 실행 (source .venv/bin/activate 실행 후)
# Airflow 3.X 버전에선 `db init`이나 `db upgrade`가 아닌 `db migrate`를 사용
$ airflow db migrate
```

## 2.3 Airflow

# Airflow 시작하기 – (1) 기본 설정

---

```
# JWT token 생성  
# 아래 명령어로 생성되는 값을 복사  
$ openssl rand -base64 16  
  
# Airflow 설정에 값 추가  
$ code ${AIRFLOW_HOME}/airflow.cfg  
  
# jwt_secret = "YOUR_JWT_SECRET_TOKEN"  
# simple_auth_manager_all_admins = True
```

## 2.3 Airflow

# Airflow 시작하기 – (1) 기본 설정

---

```
# 개발/테스트 목적으로는 `standalone`을 사용할 수 있음
$ airflow standalone

# 프로덕션 환경에서는 다음처럼 각 컴포넌트를 개별 터미널에서 실행
# airflow api-server
# airflow scheduler
# airflow dag-processor
```

### 2.3 Airflow

# Airflow 시작하기 – (1) 기본 설정

문제	출력	디버그 콘솔	터미널	포트 <span>10</span>		
포트	전달된 주소	실행 중인 프로세스			표시 여부	원본
○	3000	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>			🔒 Private	자동 전달됨
●	3306	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>	/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port...		🔒 Private	자동 전달됨
○	3894	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>			🔒 Private	자동 전달됨
●	3918	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>	/vscode/bin/linux-x64/360a4e4fd251bfce169a4ddf857c7d...		🔒 Private	자동 전달됨
○	5021	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>			🔒 Private	자동 전달됨
○	5209	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>			🔒 Private	자동 전달됨
●	8080	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>	/workspaces/lgcns-advanced-mlops/.venv/bin/python3 -c fr...		🔒 Private	자동 전달됨
●	8089	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>	/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port...		🔒 Private	자동 전달됨
●	8793	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>	gunicorn: worker [gunicorn] (6662)		🔒 Private	자동 전달됨
●	8794	<a href="https://verbose-space-barn...">https://verbose-space-barn...</a>	gunicorn: worker [gunicorn] (6628)		🔒 Private	자동 전달됨
<a href="#">포트 추가</a>						

## 2.3 Airflow

# Airflow 시작하기 – (1) 기본 설정

The screenshot shows the Airflow Web UI Home page. On the left is a sidebar with icons for Home, Dags, Assets, Browse, Admin, Docs, and User. The main area has a "Welcome" header. It includes sections for Stats (Failed dags: 0, Dag import errors: 4, Running dags: 0, Active dags: 0), Health (MetaDatabase, Scheduler, Triggerer, Dag Processor all green), History (Last 24 hours: 2025-08-19, 20:58:32 - 2025-08-20, 20:58:32), and Asset Events (0 events found). The central part displays DAG Runs (Queued: 0, Running: 0, Success: 0, Failed: 0) and Task Instances (0).

## 2.3 Airflow

# Airflow 시작하기 – (2) 프로젝트 폴더 심볼릭 링크 생성

---

```
# Airflow 기본 폴더 내 DAGS 폴더 생성
$ mkdir -p ~/airflow/dags

# 프로젝트 내 파이프라인 DAG이 저장될 폴더의 심볼릭 링크를 위 폴더에 생성
# 심볼릭 링크 = 바로가기
$ ln -snf /workspaces/advanced-mlops/utils ~/airflow/dags/utils
$ ln -snf /workspaces/advanced-mlops/pipelines ~/airflow/dags/pipelines

# 생성 확인
$ ll ~/airflow/dags
```

## 2.3 Airflow

# Airflow 시작하기 – (3) Airflow 변수 관리

The screenshot shows the Airflow UI with a sidebar on the left containing icons for Assets, Browse, and Admin. The Admin icon is highlighted with a red box. The main content area displays a timeline from "Last 24 hours" to "2025-08-19, 21:01:21 - 2025-08-20, 21:01:21". Below the timeline, there are tabs for "Variables" and "Runs", with "Variables" being the active tab. Other tabs include Pools, Providers, Plugins, Connections, and Config. Under the Variables tab, there is a section for "Success" with a green bar and a "0 Failed" message. A large black arrow points downwards from this screenshot to the next one.

The screenshot shows the Airflow Variables management page. At the top, there is a search bar with "Search Keys" and an "Advanced Search" button. Below the search bar are two buttons: "Import Variables" with an upward arrow icon and "Add Variable" with a plus sign icon, both of which are highlighted with a red box. A message "No Variables found." is displayed at the bottom.

## 2.3 Airflow

# Airflow 시작하기 – (3) Airflow 변수 관리

### Add Variable

Key \*

Value \*

Description

Reset

Save

Search Keys Advanced Search ⌘+K

Import Variables Add Variable

Key	Value	Description	Is Encrypted	Actions
AIRFLOW_DAGS_PATH	/workspaces/advanced-mlops	false		

## 2.3 Airflow

# 간단한 DAG 만들어보기

pipelines/tutorial/first\_dag.py

```
with DAG(  
    [REDACTED] → DAG에 대한 여러 파라미터를 작성  
) as dag:  
    task1 = BashOperator(  
        [REDACTED]  
    )  
    task2 = BashOperator(  
        [REDACTED] → Task를 정의하는 Operator에 맞는 파라미터를 추가  
    )  
    Task3 = BashOperator(  
        [REDACTED]  
    )  
  
    task1 >> [task2, task3] → Task를 DAG 형태로 선언해주면 끝
```

## 2.3 Airflow

# 간단한 DAG 만들어보기

`pipelines/tutorial/first_dag.py`

```
with DAG(
    dag_id="simple_dag", •————→ DAG 이름
    default_args={
        "owner": "user", •————→ DAG 관리 주체 (사용자 이름, 팀 이름 등)
        "depends_on_past": False, •————→ True인 경우 과거에 실패하지 않았을 경우에만 실행됨
        "email": "jaeyoon.han@lgcns.com",
        "email_on_failure": False, •————→ 실패 시 이메일 알림 여부
        "email_on_retry": False, •————→ 실패 시 재실행 여부
        "retries": 1,
        "retry_delay": timedelta(minutes=5),
        "on_failure_callback": failure_callback, •————→ 실패 시 호출되는 콜백 함수
        "on_success_callback": success_callback, •————→ 성공 시 호출되는 콜백 함수
    },
    description="Simple airflow dag", •————→ DAG에 대한 설명
    schedule="0 15 * * *", •————→ DAG 실행에 대한 스케줄링 (cron 문자열, timedelta 객체, Timetable 등의 객체 사용 가능)
    start_date=datetime(2025, 3, 1, tzinfo=local_timezone), •————→ DAG 실행을 시작할 날짜
    catchup=False, •————→ 스케줄러 캐치업 여부 (활성화 시 위 시작 날짜부터 현재 날짜까지 DAG가 실행됨)
    tags=set(["lgcns", "mlops"]), •————→ 태그 (필터링에 유용하며 본 실습에서는 동일한 태그 사용 예정)
) as dag:
```

## 2.3 Airflow

# 간단한 DAG 만들어보기

```

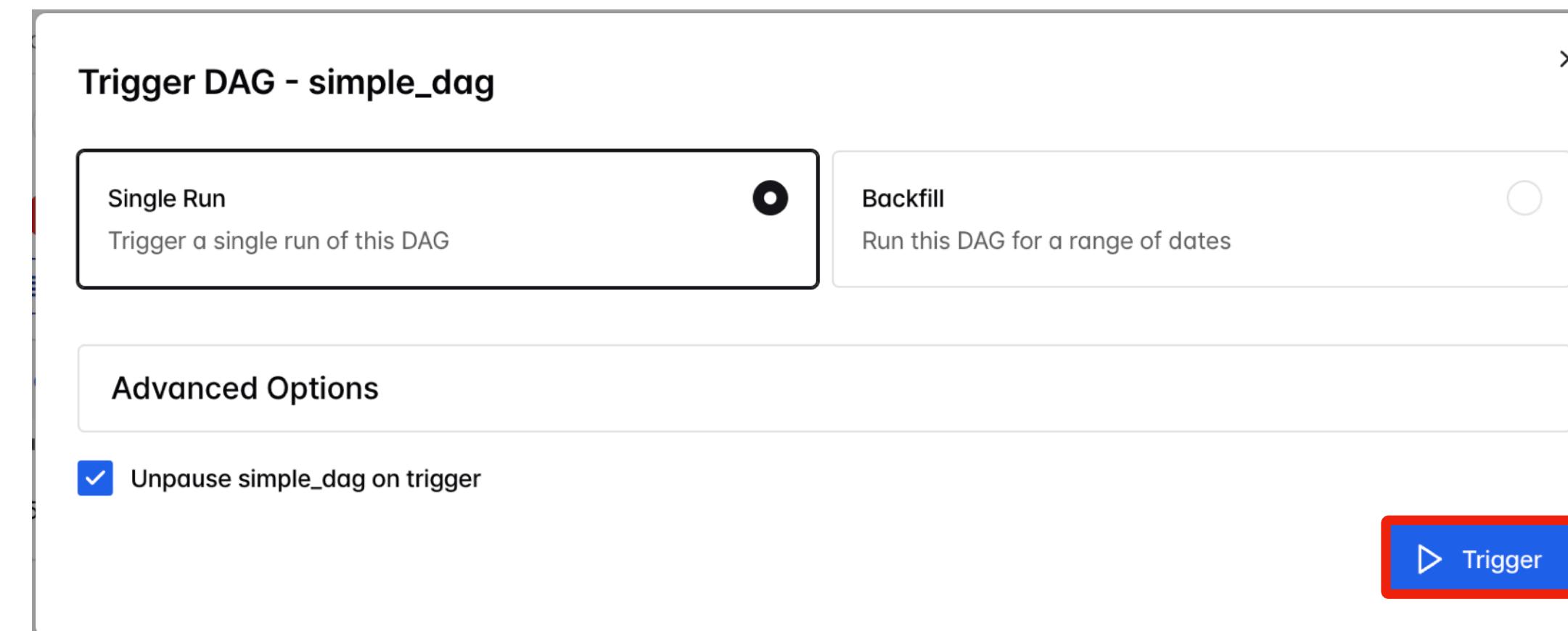
task1 = BashOperator( •————→ 각 Task 정의를 위해 Operator 사용           pipelines/tutorial/first_dag.py
    task_id="print_date",
    bash_command="date",
)
task2 = BashOperator(
    task_id="sleep",
    depends_on_past=False,
    bash_command="sleep 5",
    retries=3,
)
loop_command = dedent(
"""
{% set kst_ds = data_interval_start.in_timezone('Asia/Seoul').to_date_string() %}
{% for i in range(5) %}
    echo "ds = {{ kst_ds }}" •————→ Jinja Template 이용해서 날짜 추가 가능 (KST 설정)
    echo "macros.ds_add(ds, {{ i }}) = {{ macros.ds_add(kst_ds, i) }}"
{% endfor %}
"""
)
task3 = BashOperator(
    task_id="print_with_loop",
    bash_command=loop_command,
)
task1 >> [task2, task3] •————→ DAG 정의 (task1 → task2 or task1 → task3)

```

## 2.3 Airflow

# 간단한 DAG 만들어보기

The screenshot shows the Airflow web interface. On the left is a sidebar with icons for Home, DAGs (selected), Assets, and Browse. The main area has tabs for Dags, Runs, and Task Instances, with Dags selected. A search bar at the top has 'Igcns' entered. Below the search are filters for Failed, Running, Success, and a dropdown for 'Igcns'. A red box highlights the 'Igcns' filter. To the right is a button to 'Reset 1 filter'. A dropdown menu says 'Sort by Display Name (A-Z)'. Below this, a card for 'simple\_dag' shows it's scheduled to run at '0 15 \* \* \*' and its latest run was on '2025-08-20, 15:00:00'. A red box highlights the blue 'Trigger' button next to the DAG name.



## 2.3 Airflow

# 간단한 DAG 만들어보기

**simple\_dag** ↗ lgcns, mlops

Schedule      Latest Run      Next Run

0 15 \* \* \*      2025-08-21, 15:00:00 (✓)      2025-08-22, 15:00:00

Dag simple\_dag / Dag Run 2025-08-21, 15:00:00

Options 14s  
11s  
7s  
0s

2025-08-21, 15:00:00 ✓ success

Logical Date      Run Type      Start      End      Duration      Dag Version(s)  
2025-08-21, 15:00:00      scheduled      2025-08-22, 06:23:57      2025-08-22, 06:24:11      00:00:14      v1

Add a note      Clear Run      Mark Run as...

Task Instances      Audit Logs      Code      Details      Asset Events

Search Tasks      All States

Task ID	Map Index	State	Start Date	End Date	Try Num
print_with_loop		✓ success	2025-08-22, 06:24:11	2025-08-22, 06:24:11	1
sleep		✓ success	2025-08-22, 06:24:05	2025-08-22, 06:24:11	1
print_date		✓ success	2025-08-22, 06:23:58	2025-08-22, 06:24:05	1

## 2.3 Airflow

# 간단한 DAG 만들어보기

2025-08-21, 15:00:00 ✓ success

Add a note Clear Run Mark Run as... ▾

Logical Date	Run Type	Start	End	Duration	Dag Version(s)
2025-08-21, 15:00:00	⌚ scheduled	2025-08-22, 06:23:57	2025-08-22, 06:24:11	00:00:14	v1

Task Instances Audit Logs Code Details Asset Events

State	✓ success	
Run ID	scheduled_2025-08-21T06:00:00+00:00	
Run Type	⌚ scheduled	
Run Duration	00:00:14	
Last Scheduling Decision	2025-08-22, 06:24:11	
Queued at	2025-08-22, 06:23:57	
Start Date	2025-08-22, 06:23:57	
End Date	2025-08-22, 06:24:11	
Data Interval Start	2025-08-21, 15:00:00	
Data Interval End	2025-08-21, 15:00:00	
Trigger Source	timetable	
Version ID	0198c219-adfd-70b1-bac0-4fe521b886ae	
Dag Version(s)	Bundle Name	dags-folder
Created At	2025-08-19 20:31:59	



# M3. MLOps 파이프라인 개발 (CT)

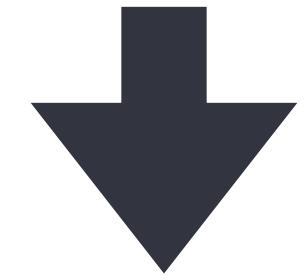
1. 데이터 추출
2. 데이터 전처리
3. 모델 학습/평가

## **3.1 데이터 추출**

### 3.1 데이터 추출

## Airflow 추가 설정

The screenshot shows the Airflow Admin interface. On the left, there is a sidebar with icons for Assets, Browse, and Admin. The Admin icon is highlighted with a red box. Below the sidebar, there is a 'History' section with a date range selector showing 'Last 24 hours' and the date range '2025-08-19, 21:01:21 - 2025-08-20, 21:01:21'. To the right of the sidebar, there is a 'Runs' section with tabs for 'Variables', 'Pools', 'Providers', 'Plugins', and 'Connections'. The 'Connections' tab is also highlighted with a red box. Below these tabs, there are sections for 'Config', 'Success' (with 0 items), and 'Failed' (with 0 items). Each section has a progress bar indicating 0% completion.



The screenshot shows the Airflow Connections page. It displays the message 'No Connections found.' and features a prominent blue button with a plus sign and the text 'Add Connection'.

### 3.1 데이터 추출

## Airflow 추가 설정

Add Connection

Connection ID \*

MySQL

Connection type missing? Make sure you have installed the corresponding Airflow Providers Package.

Standard Fields

Description

Host

Login

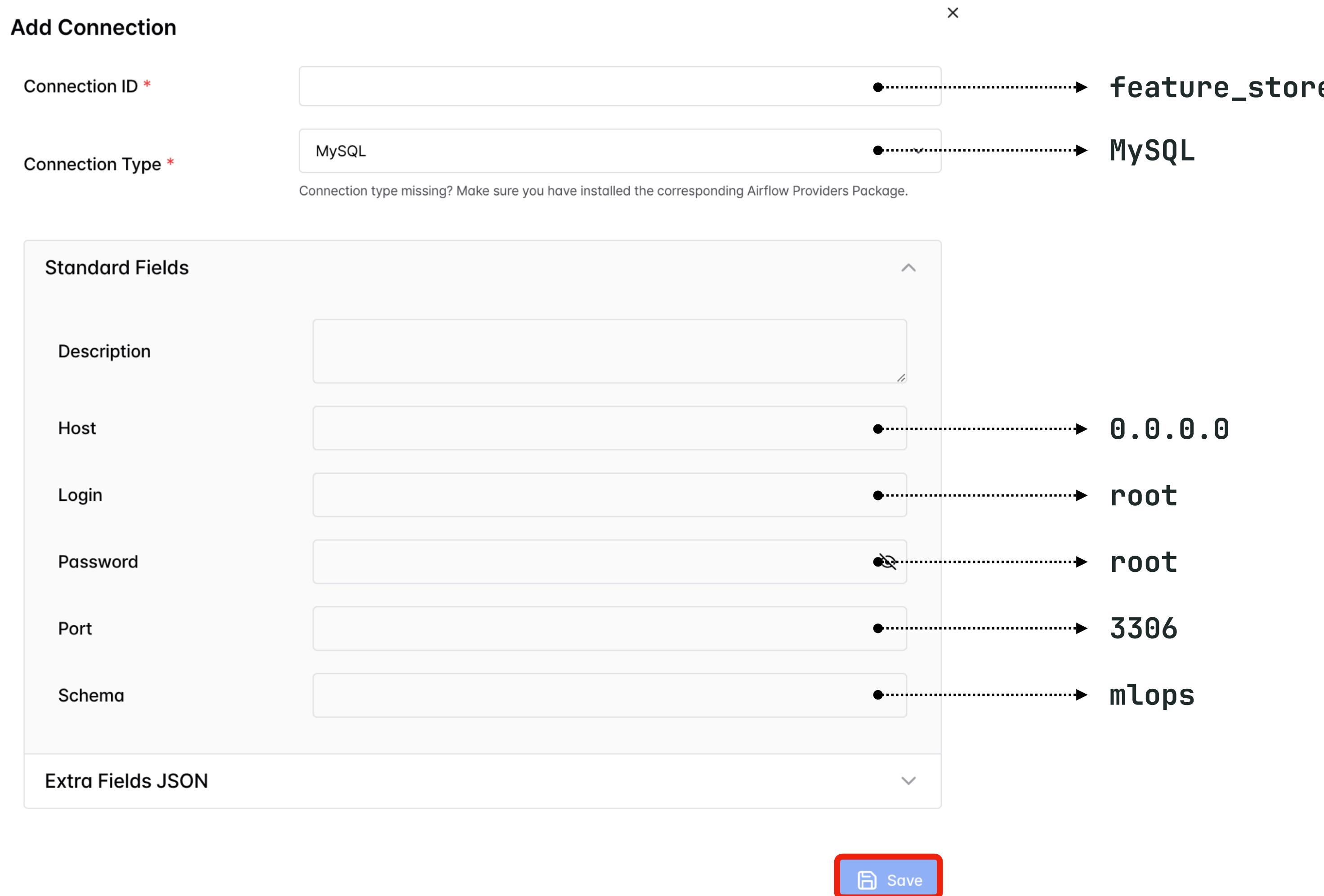
Password

Port

Schema

Extra Fields JSON

 Save



### 3.1 데이터 추출

## 데이터 병합

데이터베이스: mlops

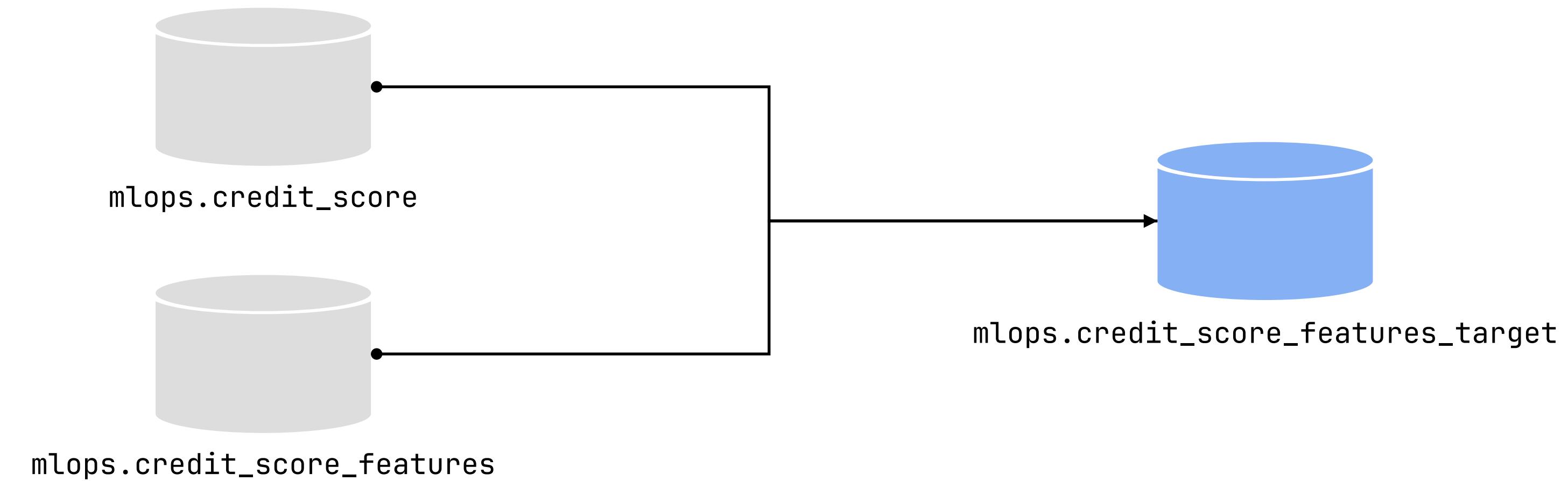
데이터베이스 변경 데이터베이스 구조 권한

테이블과 뷰

테이블 내 데이터 검색 (4)

검색

	테이블	엔진?	정렬?	데이터 길이?	색인 길이?	데이터 여유?	자동 증가?	행?	주석?
<input type="checkbox"/>	credit_score	InnoDB	utf8mb4_general_ci	131,072	0	4,194,304		~ 8	
<input type="checkbox"/>	credit_score_features	InnoDB	utf8mb4_general_ci	131,072	0	16,777,216		~ 8	
<input type="checkbox"/>	credit_score_features_target	InnoDB	utf8mb4_general_ci	868,352	0	0		~ 2,597	
<input type="checkbox"/>	customer_info	InnoDB	utf8mb4_general_ci	131,072	0	4,194,304		~ 61	
<b>총 4개</b>		InnoDB	utf8mb4_general_ci	1,261,568	0	0			



- 두 개의 테이블을 병합하여 최종적으로 **인덱스와 피처, 타겟값만 남은 피처-타겟 테이블에 데이터 추가**
  - 매일마다 생성한 데이터는 **base\_dt**라는 컬럼에 **생성 날짜를 기록**
  - 피처-타겟 테이블은 **최근 7일 데이터만 관리**하고 **이전 데이터는 매번 데이터 추가 시 삭제** 필요
  - 만약 오늘 날짜의 데이터가 있다면 해당 데이터를 지우고 다시 추가

### 3.1 데이터 추출

## 데이터 병합

---

pipelines/continuous\_training/data\_extract/features.sql

```
-- 1. 일주일 전 날짜 이전 데이터 삭제
DELETE FROM mlops.credit_score_features_target
WHERE base_dt <= DATE_FORMAT(
    DATE_ADD(
        STR_TO_DATE(
            '{{ data_interval_start.in_timezone("Asia/Seoul").to_date_string() }}',
            '%Y-%m-%d'
        ),
        INTERVAL -7 DAY
    ),
    '%Y-%m-%d'
) OR
OR base_dt = STR_TO_DATE(
    '{{ data_interval_start.in_timezone("Asia/Seoul").to_date_string() }}',
    '%Y-%m-%d'
);
```



### 3.1 데이터 추출

## 데이터 병합

---

`pipelines/continuous_training/data_extract/features.sql`

```
-- 2. 새로운 데이터 삽입
INSERT INTO mlops.credit_score_features_target (
    .....
)
SELECT STR_TO_DATE('{{ data_interval_start.in_timezone("Asia/Seoul").to_date_string() }}', '%Y-%m-%d') AS base_dt, •————→ Jinja Template
.....
FROM mlops.credit_score_features a
INNER JOIN (
    SELECT *
    FROM mlops.credit_score
    WHERE date BETWEEN DATE_ADD(
        STR_TO_DATE('{{ data_interval_start.in_timezone("Asia/Seoul").to_date_string() }}', '%Y-%m-%d'), •————→ Jinja Template
        INTERVAL -1 MONTH
    )
    AND STR_TO_DATE('{{ data_interval_start.in_timezone("Asia/Seoul").to_date_string() }}', '%Y-%m-%d') •————→ Jinja Template
) b ON a.id = b.id
    AND a.customer_id = b.customer_id;
```

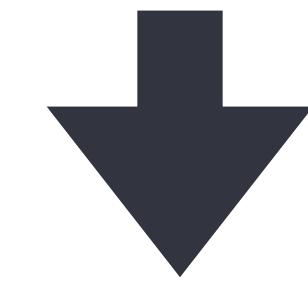
### 3.1 데이터 추출

## Task 개발

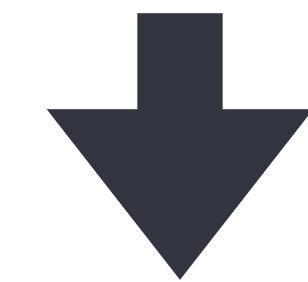
---

**EmptyOperator**

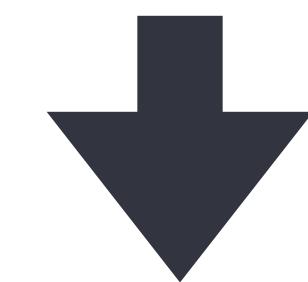
아무 작업도 하지 않는 Task로 DAG의 큰 틀을 구성



각 Task를 추가 작성



추가 작성 후 테스트



문제 없으면 다음 Task 개발

### 3.1 데이터 추출

## Task 개발

---

```

pipelines/continuous_training/continuous_training_dag.py
with DAG(
    dag_id="credit_score_classification_ct",
    default_args={
        "owner": "user",
        "depends_on_past": False,
        "email": ["otzslayer@gmail.com"],
        "on_failure_callback": failure_callback,
        "on_success_callback": success_callback,
    },
    description="A DAG for continuous training",
    schedule=None,
    start_date=datetime(2025, 1, 1, tzinfo=local_timezone),
    catchup=False,
    tags=set(["lgcns", "mlops"]),
) as dag:
    data_extract = EmptyOperator(task_id="data_extraction")  

    data_preprocessing = EmptyOperator(task_id="data_preprocessing")  

    training = EmptyOperator(task_id="model_training")  

    data_extract >> data_preprocessing >> training

```

### 3.1 데이터 추출

## Task 개발

---

pipelines/continuous\_training/continuous\_training\_dag.py

```
data_extract = SQLExecuteQueryOperator(  
    task_id="data_extraction", ● → Task 이름  
    conn_id=conn_id, ● → Airflow에서 미리 설정해놓은 Connection 이름  
    sql=read_sql_file(sql_file_path), ● → 실행할 쿼리문 (문자열이나 리스트이며, 문자열은 반드시 세미콜론(;)으로 구분되어야 함)  
    split_statements=True, ● → 여러 개의 SQL 문장을 한꺼번에 제공받았을 경우 분할 실행 여부  
)
```

### 3.1 데이터 추출

## DAG 실행 결과 확인

**Dag**  credit\_score\_classification\_ct **Dag Run** 2025-08-22, 06:24:23 **Trigger**

Options **Search Dags** **Trigger**

2025-08-22, 06:24:23 ✓ success

Add a note Clear Run Mark Run as...

Logical Date	Run Type	Start	End	Duration	Dag Version(s)
2025-08-22, 06:24:23	▶ manual	2025-08-22, 07:14:36	2025-08-22, 07:14:53	00:00:17	v1, v2

**Task Instances** Audit Logs Code Details Asset Events

Search Tasks All States

Task ID	Map Index	State	Start Date	End Date	Try Num
model_training		✓ success	2025-08-22, 07:14:47	2025-08-22, 07:14:53	2
data_preprocessing		✓ success	2025-08-22, 07:14:38	2025-08-22, 07:14:46	3
data_extraction		✓ success	2025-08-22, 07:14:36	2025-08-22, 07:14:37	5

data\_extraction data\_preprocessing model\_training

### 3.1 데이터 추출

## DAG 실행 결과 확인

The screenshot shows the Airflow web interface for monitoring DAG runs. On the left, a sidebar navigation bar includes Home, Dags (selected), Assets, Browse, Admin, Docs, and User. The main area displays the DAG run details for 'credit\_score\_classification\_ct' on 2025-08-22, 06:24:23. The 'data\_extraction' task is listed as successful (try 5) with a duration of 1.11s. The logs tab shows the following SQL code:

```

▶ Log message source details: sources=["/home/codespace/airflow/logs/dag_id=credit_score_classification_ct/run_id=manual__2025-08-21T22:14:36"]
2 [2025-08-21, 22:14:36] INFO - DAG bundles loaded: dags-folder, example_dags: source="airflow.dag_processing.bundles.DagBag"
3 [2025-08-21, 22:14:36] INFO - Filling up the DagBag from /home/codespace/airflow/dags/pipelines/continuous_training/continuous_
4 [2025-08-21, 22:14:37] INFO - Executing: -- 1. 일주일 전 날짜 이전 데이터 삭제
    DELETE FROM mlops.credit_score_features_target
    WHERE base_dt <= DATE_FORMAT(
        DATE_ADD(
            STR_TO_DATE('2025-08-22', '%Y-%m-%d'),
            INTERVAL -7 DAY
        ),
        '%Y-%m-%d'
    ) OR
    base_dt = STR_TO_DATE('2025-08-22', '%Y-%m-%d');

    -- 2. 새로운 데이터 삽입
    INSERT INTO mlops.credit_score_features_target (
        base_dt,
        id,
        customer_id,
        date,
        age,
        occupation,
        annual_income,
        monthly_inhand_salary,
        num_bank_accounts,
        ...
    )

```

## **3.2 데이터 전처리**

### 3.2 데이터 전처리

## 전처리 클래스 개발 로직

---

1

### 데이터 불러오기

- 만약 불러온 데이터가 비어있다면 오류 발생시켜야 함

2

### 수치형 변수에 대해 RobustScaler 적용

- 추후 서빙 시 각 수치형 변수에 대한 RobustScaler를 불러와 적용하기 때문에 객체 덤프 필요하며 아티팩트 폴더에 저장
- 학습/검증 데이터에 대해 적합시키고 변환 수행

3

### 변환한 학습 데이터 저장

- 아티팩트 폴더에 저장

4

### 파일 실행을 위한 인자 파서(argument parser) 설정

- 모델 이름과 실행 날짜를 인자로 받고, 이 값을 이용해 아티팩트 폴더 아래에서 관리하도록 함

### 3.2 데이터 전처리

## 전처리 클래스 개발

---

```
pipelines/continuous_training/data_preprocessing/preprocessor.py
```

# TODO 부분 코드 작성

### 3.2 데이터 전처리

## 환경 변수 관리 방안

---

### 개발 환경에서는...

- .env 파일을 이용해서 쉽게 관리할 수 있음
- 우선 프로젝트 루트 폴더에 .env 파일을 생성하고 환경변수를 작성
- 파이썬에서는 python-dotenv 라이브러리를 설치하고 `load_dotenv()` 함수로 환경변수를 불러올 수 있음

**.ENV**

```
FEATURE_STORE_URL=mysql+pymysql://root:root@localhost:3306/mlops
ARTIFACTS_PATH=/home/codespace/airflow/artifacts
```

### 운영 환경에서는...

- 개발 환경처럼 .env 파일을 사용하면 보안에 취약함
  - 특히 **Git에서 추적하게 설정했다면** 모두가 저장소에 접근하여 환경변수를 확인할 수 있음
- Airflow를 사용하는 경우에는 Variables나 Secrets에서 관리할 수 있음
- sudo 권한이 있는 경우 /etc/environment에 환경 변수를 추가하여 시스템 환경변수로 추가할 수 있음 (Linux)
- Pydantic을 활용하여 환경 설정하는 것이 바람직함

### 3.2 데이터 전처리

## Docker 설정 파일 개발

- pipelines/continuous\_training 디렉토리에 docker 라는 폴더로 이동
- 폴더 내에 **requirements.txt**, **Dockerfile**, **docker-compose.yml** 파일 설정
  - 이 세 개의 파일을 이용해 전처리 모듈, 학습/평가 모듈을 모두 컨테이너로 띄울 수 있음

```

drwxr-xr-x@   - jayhan 5 Feb 10:41 pipelines
drwxr-xr-x@   - jayhan 5 Feb 10:41 continuous_deployment
drwxr-xr-x@   - jayhan 7 Feb 10:40 continuous_training
|   └── data_extract
|       └── __init__.py
|       └── features.sql
drwxr-xr-x@   - jayhan 6 Feb 10:40 data_preprocessing
|   └── __init__.py
|   └── preprocessor.py
drwxr-xr-x@   - jayhan 7 Feb 10:41 docker
|   └── Dockerfile
|   └── docker-compose.yml
|   └── requirements.txt
drwxr-xr-x@   - jayhan 5 Feb 10:41 training
|   └── __init__.py
|   └── continuous_training_dag.py
drwxr-xr-x@   - jayhan 4 Feb 17:17 tutorial
|   └── __init__.py
|   └── first_dag.py
drwxr-xr-x@   - jayhan 4 Feb 17:17
|   └── __init__.py

```

The diagram illustrates the file structure of the `continuous_training` directory. A dashed box encloses the `docker` folder. Three arrows point from the files within this folder to their respective descriptions:

- `Dockerfile` → Dockerfile
- `docker-compose.yml` → 위 Dockerfile을 이미지로 하는 컨테이너를 띄우기 위한 설정 파일
- `requirements.txt` → 컨테이너 내에서 pip로 설치할 라이브러리 목록

### 3.2 데이터 전처리

## Docker 설정 파일 개발

### 1 Dockerfile

- 컨테이너의 기본 환경을 구성하기 위해 작성
- 엄밀하게는 개별 컨테이너의 이미지 생성 역할로 사용

```
FROM python:3.12-slim-bookworm
LABEL maintainer="otzslayer@gmail.com"
COPY --from=ghcr.io/astral-sh/uv:latest /uv /uvx /bin/

ARG USER_HOME=/home/codespace
ARG UTIL_PATH=utils
ARG PREPROCESSING_PATH=pipelines/continuous_training/data_preprocessing
ARG REQUIREMENTS_PATH=pipelines/continuous_training/docker

RUN groupadd --gid 1000 codespace \
    && useradd --uid 1000 -g codespace --shell /bin/bash --create-home codespace

COPY --chown=codespace:codespace ${UTIL_PATH}/ ${USER_HOME}/utils
COPY --chown=codespace:codespace ${PREPROCESSING_PATH}/preprocessor.py \
    ${USER_HOME}/data_preprocessing/
COPY --chown=codespace:codespace ${REQUIREMENTS_PATH}/requirements.txt \
    ${USER_HOME}/

USER codespace
WORKDIR ${USER_HOME}

ENV PYTHONUNBUFFERED=1 \
    VIRTUAL_ENV="${USER_HOME}/.venv"

ENV PATH="${VIRTUAL_ENV}/bin:$PATH"

RUN mkdir -p ${USER_HOME}/artifacts \
    && uv init --python 3.12 \
    && uv venv --python 3.12 --seed \
    && uv pip install -r ${USER_HOME}/requirements.txt
```

pipelines/continuous\_training/docker/Dockerfile

<code>drwxr-xr-x@</code>	-	<code>jayhan</code>	8 Feb 13:04	<code>data_preprocessing</code>
<code>drwxr-xr-x@</code>	-	<code>jayhan</code>	8 Feb 13:04	<code>preprocessor.py</code>
<code>.rw-r--r--@</code>	<b>9.9k</b>	<code>jayhan</code>	8 Feb 13:04	
<code>drwxr-xr-x@</code>	-	<code>jayhan</code>	8 Feb 13:04	<code>utils</code>
<code>.rw-r--r--@</code>	0	<code>jayhan</code>	8 Feb 13:04	<code>__init__.py</code>
<code>.rw-r--r--@</code>	194	<code>jayhan</code>	8 Feb 13:04	<code>callbacks.py</code>
<code>.rw-r--r--@</code>	156	<code>jayhan</code>	8 Feb 13:04	<code>common.py</code>
<code>.rw-r--r--@</code>	872	<code>jayhan</code>	8 Feb 13:04	<code>dates.py</code>
<code>.rw-r--r--@</code>	117	<code>jayhan</code>	8 Feb 13:04	<code>requirements.txt</code>

### 3.2 데이터 전처리

## Docker 설정 파일 개발

### 2 docker-compose.yml

- Dockerfile을 사용해 이미지를 빌드하고 실행
- 디렉토리 마운트, 네트워크 설정, 환경변수 설정 등 부가적인 설정 수행

```
pipelines/continuous_training/docker/docker-compose.yml
services:
  continuous_training_pipeline:
    build:
      context: ../../..
      dockerfile: pipelines/continuous_training/docker/Dockerfile
    image: credit_score_classification:ct-pipeline-latest
    container_name: credit_score_classification_ct_pipeline
    volumes:
      - ${HOME}/airflow/artifacts:/home/codespace/artifacts
    environment:
      PYTHONPATH: /home/codespace
      ARTIFACTS_PATH: /home/codespace/artifacts
      FEATURE_STORE_URL: mysql+pymysql://root:root@mariadb:3306/mllops
    command: >
      python ${PYTHON_FILE} --model_name ${MODEL_NAME} --base_dt ${BASE_DT}
  networks:
    mllops_network:
networks:
  mllops_network:
    name: mllops_network
    external: true
```

### 3.2 데이터 전처리

## Task 개발

---

`pipelines/continuous_training/continuous_training_dag.py`

```

from airflow.providers.standard.operators.bash import BashOperator

kst_ds_template = (
    "{{ data_interval_start.in_timezone('Asia/Seoul').to_date_string() }}"
) → KST 설정값 선언

data_preprocessing = BashOperator(
    task_id="data_preprocessing",
    bash_command=f"cd {airflow_dags_path}/pipelines/continuous_training/docker &&" → Bash에서 실행할 명령어 (Docker 폴더로 이동하여 이미지로 빌드 후 컨테이너 띄우기)
        "docker compose up --build && docker compose down",
    env={ → 컨테이너 띄울 때 실행할 명령어에 사용할 인자 (일종의 환경변수)
        "PYTHON_FILE": "/home/codespace/data_preprocessing/preprocessor.py",
        "MODEL_NAME": "credit_score_classification",
        "BASE_DT": kst_ds_template,
    },
    append_env=True, → 위 값을 기준 환경변수를 대체하지 않고 추가
    retries=1,
)

```

### 3.2 데이터 전처리

## Task 개발

---

### 폴더 권한에 대한 확인 필요

- 지금까지 개발한 내용을 토대로 그대로 DAG를 실행시킨다면 컨테이너에서 root 권한으로 관련 폴더를 생성하게 됨
  - 그렇다면 로컬(Codespace)에서는 권한이 없어 접근할 수 없게 됨
  - 따라서 로컬에서 먼저 관련 폴더를 생성한 다음 DAG를 실행하는 것이 적절함

```
# 실습에 사용할 폴더를 미리 생성
$ mkdir ~/airflow/artifacts
$ mkdir ~/bentoml
$ mkdir ~/mlruns

# 만약 이미 DAG를 실행해서 root 권한으로 생성된 폴더의 소유 권한을 변경해주려면
$ sudo chown codespace:codespace -R ~/airflow/artifacts
$ sudo chown codespace:codespace -R ~/bentoml
$ sudo chown codespace:codespace -R ~/mlruns
```

### 3.2 데이터 전처리

## DAG 실행 결과 확인

The screenshot shows the Airflow web interface for monitoring DAG runs. On the left, a sidebar navigation bar includes Home, Dags (selected), Assets, Browse, Admin, Docs, and User. The main area displays the 'Dag Run' details for 'credit\_score\_classification\_ct' on '2025-08-22, 06:24:23'. The 'Task' section shows the 'data\_preprocessing' task, which completed successfully at 07:14:38. The task's duration was 8.44s and it used the 'BashOperator'. Below this, the 'Logs' tab is active, showing log entries from task tries 0 to 3. The logs detail the DAG loading process, Dockerfile building, and the final successful task execution.

Operator	Try Number	Start	End	Duration	DAG Version
BashOperator	3	2025-08-22, 07:14:38	2025-08-22, 07:14:46	8.44s	v2

**Logs** Tab Content:

```

▶ Log message source details: sources=["/home/codespace/airflow/logs/dag_id=credit_score_classification_ct/run_id=manual_2025-08-21T22:14:38"]
  2 [2025-08-21, 22:14:38] INFO - DAG bundles loaded: dags-folder, example_dags: source="airflow.dag_processing.bundles.manager.DagBag"
  3 [2025-08-21, 22:14:38] INFO - Filling up the DagBag from /home/codespace/airflow/dags/pipelines/continuous_training/continuous_
  4 [2025-08-21, 22:14:38] INFO - Tmp dir root location: /tmp: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  5 [2025-08-21, 22:14:38] INFO - Running command: ['/usr/bin/bash', '-c', 'cd /workspaces/lgcns-advanced-mlops/pipelines/continuous_
  6 [2025-08-21, 22:14:38] INFO - Output:: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  7 [2025-08-21, 22:14:38] INFO - time="2025-08-22T07:14:38+09:00" level=warning msg="COMPOSE_BAKE=false is deprecated, support for
  8 [2025-08-21, 22:14:38] INFO - #0 building with "default" instance using docker driver: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  9 [2025-08-21, 22:14:38] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  10 [2025-08-21, 22:14:38] INFO - #1 [continuous_training_pipeline internal] load build definition from Dockerfile: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  11 [2025-08-21, 22:14:38] INFO - #1 transferring dockerfile: 1.18kB done: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  12 [2025-08-21, 22:14:38] INFO - #1 DONE 0.0s: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  13 [2025-08-21, 22:14:38] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  14 [2025-08-21, 22:14:38] INFO - #2 [continuous_training_pipeline internal] load metadata for docker.io/library/python:3.12-slim-bionic: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  15 [2025-08-21, 22:14:39] INFO - #2 ...: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  16 [2025-08-21, 22:14:39] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  17 [2025-08-21, 22:14:39] INFO - #3 [continuous_training_pipeline auth] astral-sh/uv:pull token for ghcr.io: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  18 [2025-08-21, 22:14:39] INFO - #3 DONE 0.0s: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  19 [2025-08-21, 22:14:39] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  20 [2025-08-21, 22:14:39] INFO - #4 [continuous_training_pipeline internal] load metadata for ghcr.io/astral-sh/uv:latest: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  21 [2025-08-21, 22:14:39] INFO - #4 ...: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  22 [2025-08-21, 22:14:39] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  23 [2025-08-21, 22:14:39] INFO - #5 [continuous_training_pipeline auth] library/python:pull token for registry-1.docker.io: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  24 [2025-08-21, 22:14:39] INFO - #5 DONE 0.0s: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  25 [2025-08-21, 22:14:39] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
  26 [2025-08-21, 22:14:39] INFO - #4 [continuous_training_pipeline internal] load metadata for ghcr.io/astral-sh/uv:latest: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocessHook"
  
```

### **3.3 모델 학습/평가**

### 3.3 모델 학습/평가

## 모델 학습/평가 클래스 개발 로직

---

### 데이터 불러오기

1

- 불러온 후 학습 데이터와 검증 데이터에서 피처와 타겟값을 분리
- CatBoost에서 사용 가능한 형태로 변환

2

### 하이퍼파라미터 튜닝

- CatBoost 모델을 학습하며 Grid Search로 최적의 하이퍼파라미터 탐색

3

### 학습 결과를 MLflow와 아티팩트 폴더에 저장

- 튜닝 중 매 시행에 대한 메타데이터를 MLflow에 저장하고 모델을 아티팩트 폴더에 저장

4

### 최적 모델을 저장

- 추후 배포를 위해 최적 모델을 BentoML로 저장

5

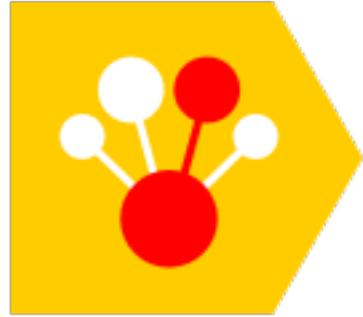
### DAG 내 Task 업데이트

- 데이터 전처리 때와 유사하게 이미지를 빌드하고 컨테이너를 띄우는 명령어로 구성된 Task 업데이트
- 일부 구성 업데이트

### 3.3 모델 학습/평가

## CatBoost

---



### CatBoost

- 러시아의 구글로 불리우는 Yandex에서 개발한 **그라디언트 부스팅(gradient boosting)** 기법
- 범주형 변수 처리에 장점을 갖는 모델

## 특징

### 다양한 피처 타입에 대한 높은 처리 성능

1

- 범주형 피처 처리
  - 자체적인 **Ordererd Boosting** 기법과 **Target Statistics(TS)** 알고리즘을 통해 범주형 피처를 정교하게 처리함
  - 데이터 누출(Data leakage)을 방지하고 과적합을 방지
- 텍스트 피처 처리
  - 자체적으로 텍스트 피처를 처리하는 알고리즘을 내장하고 있음

2

### 하이퍼파라미터 튜닝 최소화

- 다른 GBM 계열 알고리즘(XGBoost, LightGBM 등)에 비해 **하이퍼파라미터 튜닝 없이 기본값으로도 좋은 성능을 보임**

### 3.3 모델 학습/평가

## 모델 학습/평가 클래스 개발

---

`pipelines/continuous_training/training/trainer.py`

# TODO 부분 코드 작성

### 3.3 모델 학습/평가

## Docker 설정 파일 개발

### 1 Dockerfile

- 학습 관련 폴더를 복사하는 명령 추가

pipelines/continuous\_training/docker/Dockerfile

```
.....  
  
ARG REQUIREMENTS_PATH=pipelines/continuous_training/docker  
ARG TRAINING_PATH=pipelines/continuous_training/training  
  
RUN groupadd --gid 1000 codespace \  
    && useradd --uid 1000 --gid codespace --shell /bin/bash --create-home codespace  
  
COPY --chown=codespace:codespace ${UTIL_PATH}/ ${USER_HOME}/utils  
COPY --chown=codespace:codespace ${PREPROCESSING_PATH}/preprocessor.py \  
    ${USER_HOME}/data_preprocessing/  
COPY --chown=codespace:codespace ${TRAINING_PATH}/trainer.py \  
    ${USER_HOME}/training/  
COPY --chown=codespace:codespace ${REQUIREMENTS_PATH}/requirements.txt \  
    ${USER_HOME}/  
  
.....
```

### 3.3 모델 학습/평가

## Docker 설정 파일 개발

### 2 docker-compose.yml

- 컨테이너 내의 BentoML 관련 폴더를 로컬에서 마운트하도록 수정

`pipelines/continuous_training/docker/docker-compose.yml`

```
.....
volumes:
  - ${HOME}/airflow/artifacts:/home/codespace/artifacts
  - ${HOME}/bentoml:/home/codespace/bentoml
  - ${HOME}/mlruns:/home/codespace/mlruns
environment:
  PYTHONPATH: /home/codespace
  ARTIFACTS_PATH: /home/codespace/artifacts
  FEATURE_STORE_URL: mysql+pymysql://root:root@mariadb:3306/mllops
command: >
  python ${PYTHON_FILE} --model_name ${MODEL_NAME} --base_dt ${BASE_DT}
networks:
  mllops_network:
....
```

### 3.3 모델 학습/평가

## Task 개발

---

`pipelines/continuous_training/continuous_training_dag.py`

```
from airflow.providers.standard.operators.bash import BashOperator

training = BashOperator(
    task_id="model_training",
    bash_command=f"cd {airflow_dags_path}/pipelines/continuous_training/docker && "
    "docker compose up --build && docker compose down",
    env={ • → 컨테이너 띄울 때 실행할 명령어에 사용할 인자 (일종의 환경변수)
        "PYTHON_FILE": "/home/codespace/training/trainer.py",
        "MODEL_NAME": "credit_score_classification",
        "BASE_DT": kst_ds_template,
    },
    append_env=True, • → 위 값을 기준 환경변수를 대체하지 않고 추가
    retries=1,
)
```

### 3.3 모델 학습/평가

## DAG 실행 결과 확인

The screenshot shows the Airflow web interface for monitoring DAG runs. On the left, a sidebar navigation bar includes Home, Dags (selected), Assets, Browse, Admin, Docs, and User sections. The main content area displays the 'Dag Run' details for 'credit\_score\_classification\_ct' DAG, specifically for the run on 2025-08-22 at 06:24:23. The 'Task' section shows the 'model\_training' task, which has completed successfully. The task log is displayed, showing the command and its execution details. The logs indicate the DAG was loaded from a local folder, the temporary directory was set up, and the command was run successfully.

Operator	Try Number	Start	End	Duration	DAG Version
BashOperator	2	2025-08-22, 07:14:47	2025-08-22, 07:14:53	5.81s	v2

```

▶ Log message source details: sources=["/home/codespace/airflow/logs/dag_id=credit_score_classification_ct/run_id=manual__2025-08-21T22:14:47"]
 2 [2025-08-21, 22:14:47] INFO - DAG bundles loaded: dags-folder, example_dags: source="airflow.dag_processing.bundles.manager.DagBag"
 3 [2025-08-21, 22:14:47] INFO - Filling up the DagBag from /home/codespace/airflow/dags/pipelines/continuous_training/continuous_
 4 [2025-08-21, 22:14:47] INFO - Tmp dir root location: /tmp: source="airflow.task.hooks.airflow.providers.standard.hooks.subproce
 5 [2025-08-21, 22:14:47] INFO - Running command: ['/usr/bin/bash', '-c', 'cd /workspaces/lgcns-advanced-mlops/pipelines/continuu
 6 [2025-08-21, 22:14:47] INFO - Output:: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
 7 [2025-08-21, 22:14:47] INFO - time="2025-08-22T07:14:47+09:00" level=warning msg="COMPOSE_BAKE=false is deprecated, support for
 8 [2025-08-21, 22:14:47] INFO - #0 building with "default" instance using docker driver: source="airflow.task.hooks.airflow.provi
 9 [2025-08-21, 22:14:47] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
10 [2025-08-21, 22:14:47] INFO - #1 [continuous_training_pipeline internal] load build definition from Dockerfile: source="airflow
11 [2025-08-21, 22:14:47] INFO - #1 transferring dockerfile: 1.18kB done: source="airflow.task.hooks.airflow.providers.standard.h
12 [2025-08-21, 22:14:47] INFO - #1 DONE 0.0s: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessH
13 [2025-08-21, 22:14:47] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
14 [2025-08-21, 22:14:47] INFO - #2 [continuous_training_pipeline internal] load metadata for docker.io/library/python:3.12-slim-b
15 [2025-08-21, 22:14:47] INFO - #2 ...: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
16 [2025-08-21, 22:14:47] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
17 [2025-08-21, 22:14:47] INFO - #3 [continuous_training_pipeline internal] load metadata for ghcr.io/astral-sh/uv:latest: source=
18 [2025-08-21, 22:14:47] INFO - #3 DONE 0.3s: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessH
19 [2025-08-21, 22:14:47] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
20 [2025-08-21, 22:14:47] INFO - #2 [continuous_training_pipeline internal] load metadata for docker.io/library/python:3.12-slim-b
21 [2025-08-21, 22:14:47] INFO - #2 DONE 0.3s: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessH
22 [2025-08-21, 22:14:47] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"
23 [2025-08-21, 22:14:47] INFO - #4 [continuous_training_pipeline internal] load .dockergignore: source="airflow.task.hooks.airflow
24 [2025-08-21, 22:14:47] INFO - #4 transferring context: 2B done: source="airflow.task.hooks.airflow.providers.standard.hooks.su
25 [2025-08-21, 22:14:47] INFO - #4 DONE 0.0s: source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessH
26 [2025-08-21, 22:14:47] INFO - : source="airflow.task.hooks.airflow.providers.standard.hooks.subprocess.SubprocessHook"

```

verbose-space-barnacle-pj6pw5w5gw627qjx-8080.app.github.dev



# M4. MLOps 파이프라인 개발 (CD)

1. 모델 배포 (API 개발)
2. 지속적 배포 구현

## **4.1 모델 배포 (API 개발)**

#### 4.1 모델 배포 (API 개발)

## API 개발 요소

---

### 1 데이터 모델 개발

- 추론 결과를 저장할 테이블과 관련된 테이블 모델 개발
- API 입력값과 출력값의 데이터 모델 개발

### 2 아티팩트 불러오기

- 저장한 CatBoost 모델과 전처리 시 저장한 RobustScaler 불러오기

### 3 모델 추론

- 입력 받은 데이터로 추론 수행하여 결과 레이블과 그 확률값 반환

### 4 로그 데이터 적재

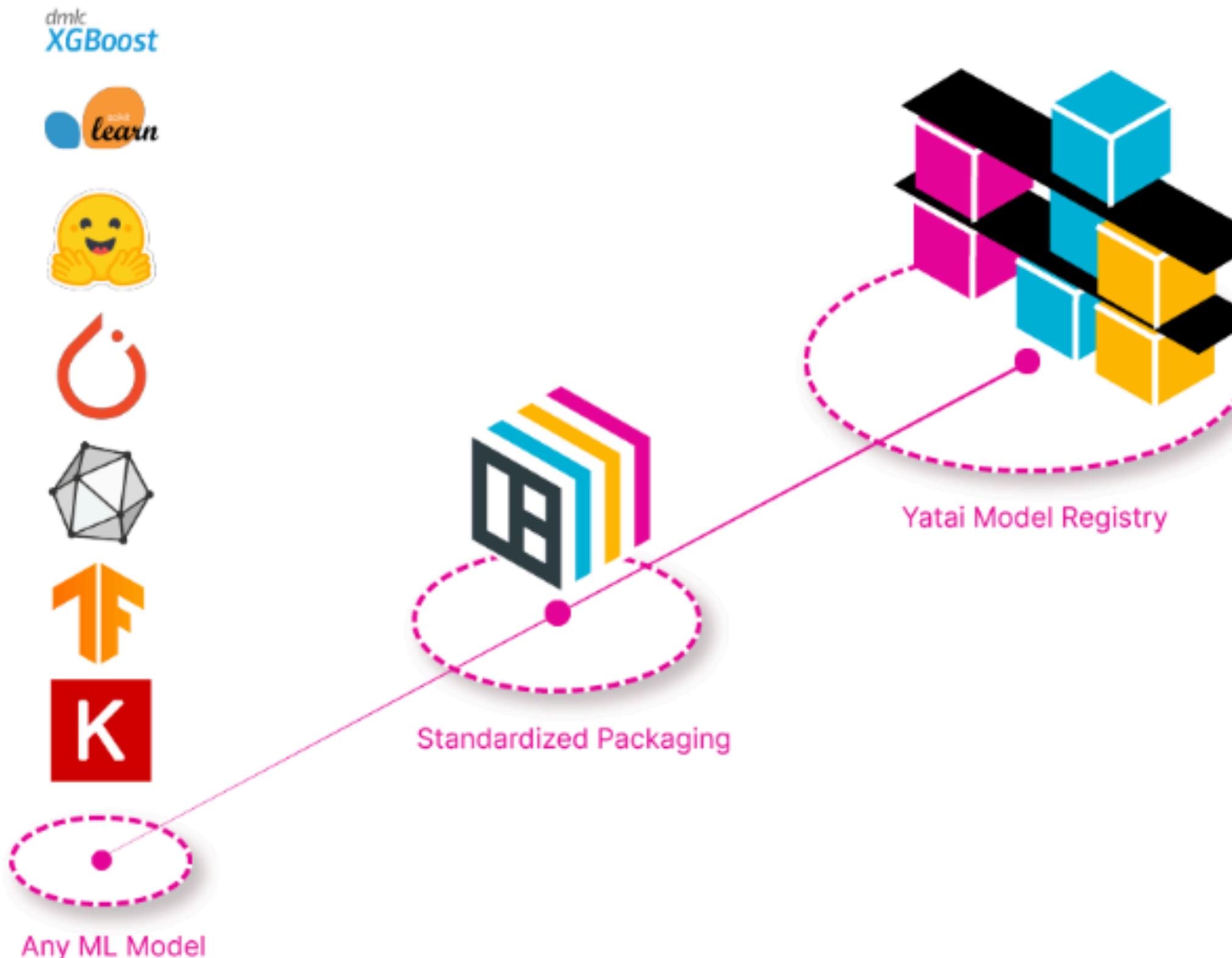
- 입력 받은 데이터, 결과 레이블, 확률값, 수행 시간 등을 생성해놓은 테이블에 적재

### 5 컨테이너 개발

- API 서비스를 띄워놓기 위한 Docker 컨테이너 개발
- DAG 개발은 진행하지 않음

#### 4.1 모델 배포 (API 개발)

## BentoML (Remind)



- **ML 모델 서빙만을 위한 라이브러리**
  - 대부분의 메이저 ML 모델 라이브러리 지원
  - 대부분의 퍼블릭 클라우드에서 사용 가능
  - 최근에는 LLM Ops도 지원함
- 코드 기반으로 이후 Airflow 등 오케스트레이션 도구를 이용하여 Task로 만들 수 있음
- 배치 추론, 실시간 추론 모두 지원함
- 웹 대시보드로 모델 관리나 API 관리 가능
- 코드 몇 줄과 커맨드 몇 줄로 손쉽게 서빙 API 구축 가능

## 4.1 모델 배포 (API 개발)

# API 폴더 구조

```

drwxr-xr-x@  - jayhan 12 Feb 00:13 api
drwxr-xr-x  - jayhan 12 Feb 00:14 docker → Docker 관련 파일 관리
.rw-r--r--  887 jayhan 12 Feb 00:50
.rw-r--r--  700 jayhan 12 Feb 00:45
drwxr-xr-x@  - jayhan 12 Feb 11:14 src
.rw-r--r--    0 jayhan 11 Feb 15:15
.rw-r--r--  555 jayhan 12 Feb 11:14 → SQLAlchemy 연동 관련 코드
.rw-r--r--  557 jayhan 12 Feb 11:39 → 테이블 모델
.rw-r--r--  1.6k jayhan 12 Feb 11:18 → API 입력값, 출력값 데이터 모델
.rw-r--r--  158 jayhan 11 Feb 18:04
.rw-r--r--  134 jayhan 12 Feb 11:49
.rw-r--r--@ 2.0k jayhan 12 Feb 13:20 → 컨테이너 띄울 때 설치할 라이브러리 목록
                                         → BentoML 서비스 코드

```

## 4.1 모델 배포 (API 개발)

# SQLAlchemy 연동

```
feature_store_url = os.getenv("FEATURE_STORE_URL")                                     api/src/db.py

[engine = create_engine(feature_store_url, echo=False)]
SessionLocal = sessionmaker(autocommit=False, autoflush=False, expire_on_commit=False, bind=engine)
Base = declarative_base()
```

## 커넥션 풀(Connection Pool)

- 데이터베이스 연결을 미리 여러 개 만들어놓고 필요할 때마다 가져가서 사용하고 반납하는 **일종의 커넥션 창고**
- 사용 이유
  - 데이터베이스에 연결하는 과정은 **생각보다 비용이 많이 들어** 애플리케이션 성능을 크게 저하함
    - 네트워크 핸드쉐이크, 인증 등
    - 연결을 수행하는 시간이 균일하지 않아 **안정적인 성능 보장이 어려움**
  - 사용 이점
    - **성능 향상**
      - : 연결 생성/종료에 드는 시간을 없애 애플리케이션의 응답 속도가 빨라짐
    - **자원 효율성**
      - : 정해진 수의 연결만 사용하므로 데이터베이스 서버의 부하를 예측하고 관리할 수 있음
    - **안정성**
      - : 커넥션 풀은 오래되어 끊어진 연결(Stale Connection)을 자동으로 감지하고 새로운 연결로 교체하는 등 연결을 안정적으로 관리해줌

## 4.1 모델 배포 (API 개발)

# SQLAlchemy 연동

```
feature_store_url = os.getenv("FEATURE_STORE_URL")

engine = create_engine(feature_store_url, echo=False)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, expire_on_commit=False, bind=engine)
Base = declarative_base()
```

api/src/db.py

## sessionmaker

- SQLAlchemy에서 데이터베이스 세션을 생성하는 팩토리
- DB와의 연결을 관리하는 세션을 쉽게 만들고 사용할 수 있도록 도와주는 도구

## sessionmaker의 필요성

1. 요청마다 독립적인 작업 공간 제공해 독립성 및 스레드 안정성을 제공
  - 각각의 API 요청이 들어올 때마다 sessionmaker를 호출해 별도의 세션을 생성
  - 각 사용자의 요청은 각자의 세션에서만 처리되므로 스레드에 안전(thread-safe)한 작업
2. 일관된 세션 생성
  - sessionmaker를 한 곳에서 설정을 관리하여 동일한 설정을 가진 세션을 반복해서 쉽게 만들 수 있음
3. 트랜잭션 관리 용이
  - 세션을 sessionmaker를 통해 생성한 경우 with session.begin() 절을 통해 명확한 구조로 트랜잭션을 관리할 수 있음

## 4.1 모델 배포 (API 개발)

# 테이블 모델

---

api/src/models.py

```
from sqlalchemy import JSON, Column, DateTime, Float, Integer, String, func
from api.src.db import Base

class CreditPredictionApiLog(Base):
    __tablename__ = "credit_predictions_api_log"

    id = Column(Integer, primary_key=True, autoincrement=True)
    customer_id = Column(String(10), nullable=False)
    features = Column(JSON, nullable=False)
    prediction = Column(String(10), nullable=False)
    confidence = Column(Float, nullable=False)
    elapsed_ms = Column(Integer, nullable=False)
    created_at = Column(DateTime, server_default=func.now())
```

## 4.1 모델 배포 (API 개발)

# 서비스 코드

---

api/services.py

# TODO 부분 코드 작성

## 4.1 모델 배포 (API 개발)

# 서비스 코드

---

api/services.py

```

@bentoml.service(●————→ 서비스될 클래스에 사용할 자원과 서비스 타임아웃 설정
    resources={"cpu": "2"},
    traffic={"timeout": 10},
)
class CreditScoreClassifier:
    def __init__(self) → None:
        self.session_maker = None ●————→ 세션 팩토리 변수 선언
        self.bento_model = bentoml.models.get("credit_score_classifier:latest")●————→ 저장한 모델 불러오기
        self.robust_scalers = joblib.load(
            os.path.join(encoder_path, "robust_scaler.joblib")●————→ 전처리 시 저장한 RobustScaler 객체 불러오기
        )
        self.model = bentoml.catboost.load_model(self.bento_model)

    @bentoml.on_startup ●————→ BentoML 서비스 시 자동으로 수행할 함수 데코레이터
    def initialize(self):
        self.session_maker = SessionLocal●————→ 세션 팩토리 설정

```

## 4.1 모델 배포 (API 개발)

# 서비스 코드

```

@bentoml.api • → API 엔드포인트 선언 및 자동으로 입력/출력값에 대한 명세 확인 api/services.py
def predict(self, data: Features) → Response: • → 입력값과 출력값에 대한 타입 힌팅
    start_time = time.time()
    df = pd.DataFrame([data.model_dump()]) • → Pydantic 데이터 모델로 들어온 데이터는 model_dump() 메서드로 변환 필요
    customer_id = df.pop("customer_id").items()

    for col, scaler in self.robust_scalers.items():
        df[col] = scaler.transform(df[[col]])

    prob = np.max(self.model.predict(df, prediction_type="Probability"))
    label = self.model.predict(df, prediction_type="Class").item()
    elapsed_ms = (time.time() - start_time) * 1000

    record = CreditPredictionApiLog(• → 로그 테이블에 저장할 정보
        customer_id=customer_id,
        features=data.model_dump(),
        prediction=label,
        confidence=prob,
        elapsed_ms=elapsed_ms,
    )
    with self.session_maker() as db: • → 세션을 생성한 다음 세션을 시작하여 로그 저장
        with db.begin():
            db.add(record)

    return Response(customer_id=customer_id, predict=label, confidence=prob) • → 출력값 형태에 맞춰 결과 반환

```

#### 4.1 모델 배포 (API 개발)

## API 서비스 실행

---

```
# api 폴더로 이동하여 서비스 실행
$ cd api
$ bentoml serve

# 에러 발생 시 다음 환경변수 추가
$ export PYTHONPATH=/workspaces/advanced-mlops
```

## 4.1 모델 배포 (API 개발)

# Swagger UI를 통해 결과 확인

None OAS 3.0

[./docs.json](#)

## CreditScoreClassifier:dev

BentoML 1.3.21 docs passing Join BentoML Slack Stars 7.3k Follow BentoML

This is a Machine Learning Service created with BentoML.

### Help

- Documentation: Learn how to use BentoML.
- Community: Join the BentoML Slack community.
- GitHub Issues: Report bugs and feature requests.
- Tip: you can also [customize this README](#).

Contact BentoML Team

Servers . ▾

### Service APIs

BentoML Service API endpoints for inference. ^

POST /predict ▾

## 4.1 모델 배포 (API 개발)

# Swagger UI를 통해 결과 확인

POST /predict

Parameters

No parameters

Request body application/json

Example Value | Schema

```
{  
  "data": {  
    "customer_id": 0,  
    "age": 0,  
    "occupation": "string",  
    "annual_income": 0,  
    "monthly_inhand_salary": 0,  
    "num_bank_accounts": 0,  
    "num_credit_card": 0,  
    "interest_rate": 0,  
    "num_of_loan": 0,  
    "type_of_loan": "string",  
    "delay_from_due_date": 0,  
    "num_of_delayed_payment": 0,  
    "changed_credit_limit": 0,  
    "num_credit_inquiries": 0,  
    "credit_mix": "string",  
    "outstanding_debt": 0,  
    "credit_utilization_ratio": 0,  
    "credit_history_age": 0,  
    "payment_of_min_amount": "string",  
    "total_emi_per_month": 0,  
    "amount_invested_monthly": 0,  
    "payment_behaviour": "string",  
    "monthly_balance": 0  
  }  
}
```

## 4.1 모델 배포 (API 개발)

# Swagger UI를 통해 결과 확인

**Responses**

**Curl**

```
curl -X 'POST' \
  'http://localhost:3000/predict' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "data": {
      "customer_id": 0,
      "age": 1,
      "occupation": "string",
      "annual_income": 0,
      "monthly_inhand_salary": 0,
      "num_bank_accounts": 0,
      "num_credit_card": 0,
      "interest_rate": 0,
      "num_of_loan": 0,
      "type_of_loan": "string",
      "delay_from_due_date": 0,
      "num_of_delayed_payment": 0,
      "changed_credit_limit": 0,
      "num_credit_inquiries": 0,
      "credit_mix": "Good",
      "outstanding_debt": 0,
      "credit_utilization_ratio": 0,
      "credit_history_age": 1,
      "payment_of_min_amount": "NM",
      "total_cpi_per_month": 0
    }
  }'
```

**Request URL**

```
http://localhost:3000/predict
```

**Server response**

Code	Details
200	<b>Response body</b> <pre>{   "customer_id": 0,   "predict": "Good",   "confidence": 0.5018301391927832 }</pre> <div style="display: flex; justify-content: space-between;"> <span></span> <span></span> </div> <b>Response headers</b> <pre>content-length: 66 content-type: application/json date: Wed, 12 Feb 2025 05:45:32 GMT server: BentoML Service/CreditScoreClassifier x-bentoml-request-id: 043c5bb5ea8e17e9</pre>

## 4.1 모델 배포 (API 개발)

# 부하 테스트

---

(MLOps 관점에서)

## 부하 테스트란?

- ML 모델 API가 실제 서비스 환경에서 사용자 요청을 얼마나 잘 처리하는지 확인하고, 성능 한계점을 파악하는 과정
- API에 최대 사용자 수를 설정하고 초당 증가 사용자 수를 설정하여 부하 확인

## 부하 테스트가 중요한 이유

-  **ML은 자원 집약적(computationally intensive)이기 때문**
  - ML 모델 추론은 **CPU, GPU, 메모리** 등 시스템 자원을 상대적으로 많이 사용
  - 요청이 조금만 몰려도 시스템 전체가 급격히 느려질 수 있음
-  **응답 시간의 민감성**
  - 추천 시스템과 같은 일부 ML 서비스는 **실시간에 가까운 응답이 필요**
  - 응답 시간의 지연이 **사용자의 이탈**로 이어질 수 있기 때문
-  **예측 불가능한 병목 현상 확인**
  - 모델 자체의 연산 속도뿐만 아니라, 데이터 전처리, 네트워크 지연, DB 조회 등 다양한 구간에서 발생할 수 있는 병목을 확인
  - 대규모 처리를 위한 **DB 비동기(async)** 처리를 하는 부분은 문제가 없나?
  - DB 연동 시간이 짧으면 **비동기 작업 자체의 오버헤드가 오히려 응답 시간을 늦출 수 있음** (예측 불가능한 병목 현상)

#### 4.1 모델 배포 (API 개발)

## Locust를 이용한 부하 테스트



- Python에서 부하 테스트를 쉽게 수행할 수 있도록 도와주는 도구
- UI를 통해서 테스트 결과, 차트 등을 확인할 수 있음

tests/locust\_load\_test.py

```
import random

from locust import HttpUser, between, task

def get_random_features(): ● → 부하 테스트에 사용할 데이터(페이로드, payload) 생성
    return {
        "customer_id": random.randint(1, 100000),
        "age": random.randint(18, 70), ● → 모든 피처에 대해 유효한 범위 내에서 값 생성
        ...
    }
```

#### 4.1 모델 배포 (API 개발)

## Locust를 이용한 부하 테스트

tests/locust\_load\_test.py

```
class CreditScoreUser(HttpUser):
    wait_time = between(1, 5) •————→ 작업 사이의 대기 시간 (1~5초 사이로 랜덤)

    @task
    def predict(self):
        headers = {"Content-Type": "application/json"}
        features = get_random_features() •————→ 랜덤값으로 이루어진 피처 생성
        payload = {"data": features}
        self.client.post("/predict", json=payload, headers=headers) •————→ 예측 API 엔드포인트 호출 (POST)
```

#### 4.1 모델 배포 (API 개발)

## Locust를 이용한 부하 테스트

---

```
# 다음 명령어로 Locust 실행 후 부하테스트 수행  
# 기본 포트가 겹치므로 8090 포트 명시  
# API 서버는 미리 띄워놓은 BentoML 서버 URL 명시  
locust -f tests/locust_load_test.py --host http://localhost:3000 --web-port 8090
```

#### 4.1 모델 배포 (API 개발)

## Locust를 이용한 부하 테스트

Start new load test

Number of users (peak concurrency)\*  
100

Ramp up (users started/second)\*  
5

Host  
http://localhost:3000

Advanced options

Run time (e.g. 20, 20s, 3m, 2h, 1h20m, 3h30m10s, etc.)  
120s

Profile

START

Host  
http://localhost:3000

Status  
READY

RPS  
0

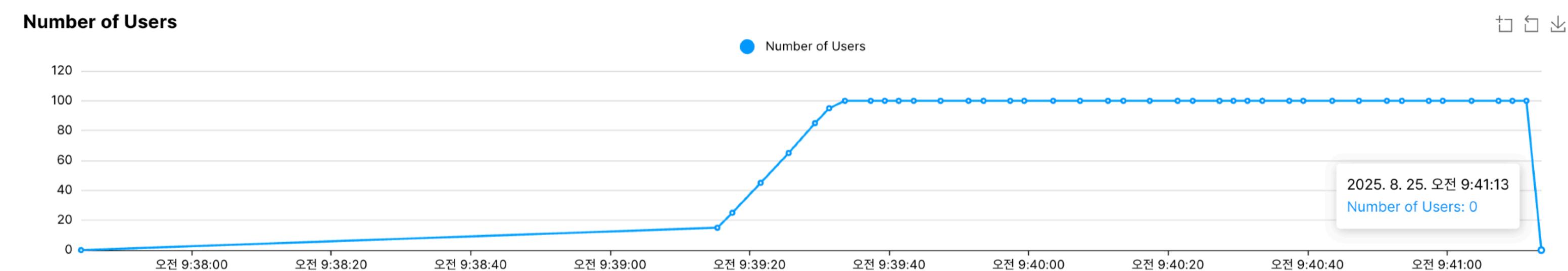
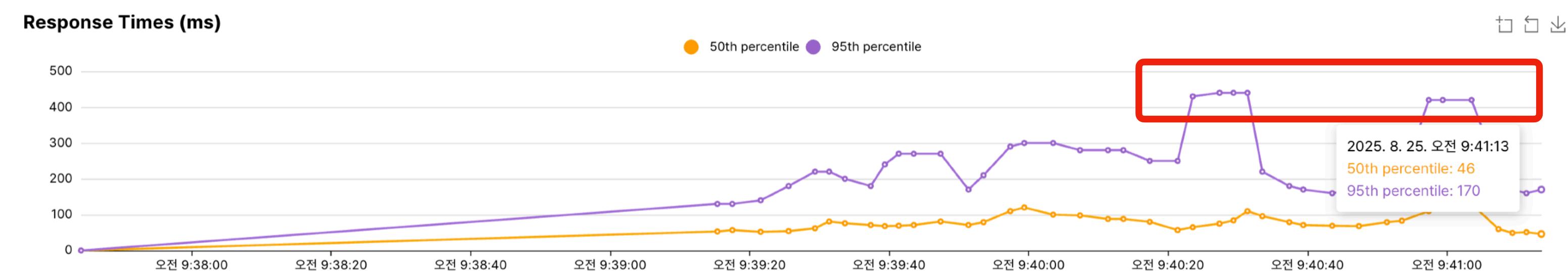
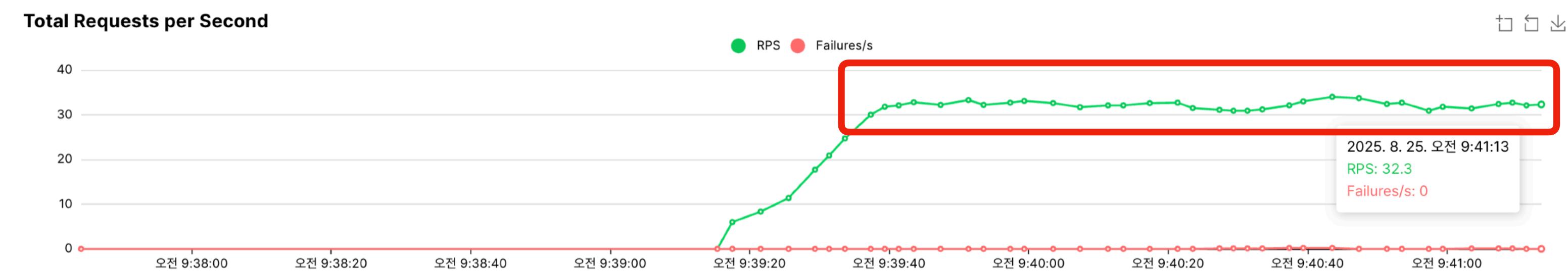
Failures  
0%

⚙️

## 4.1 모델 배포 (API 개발)

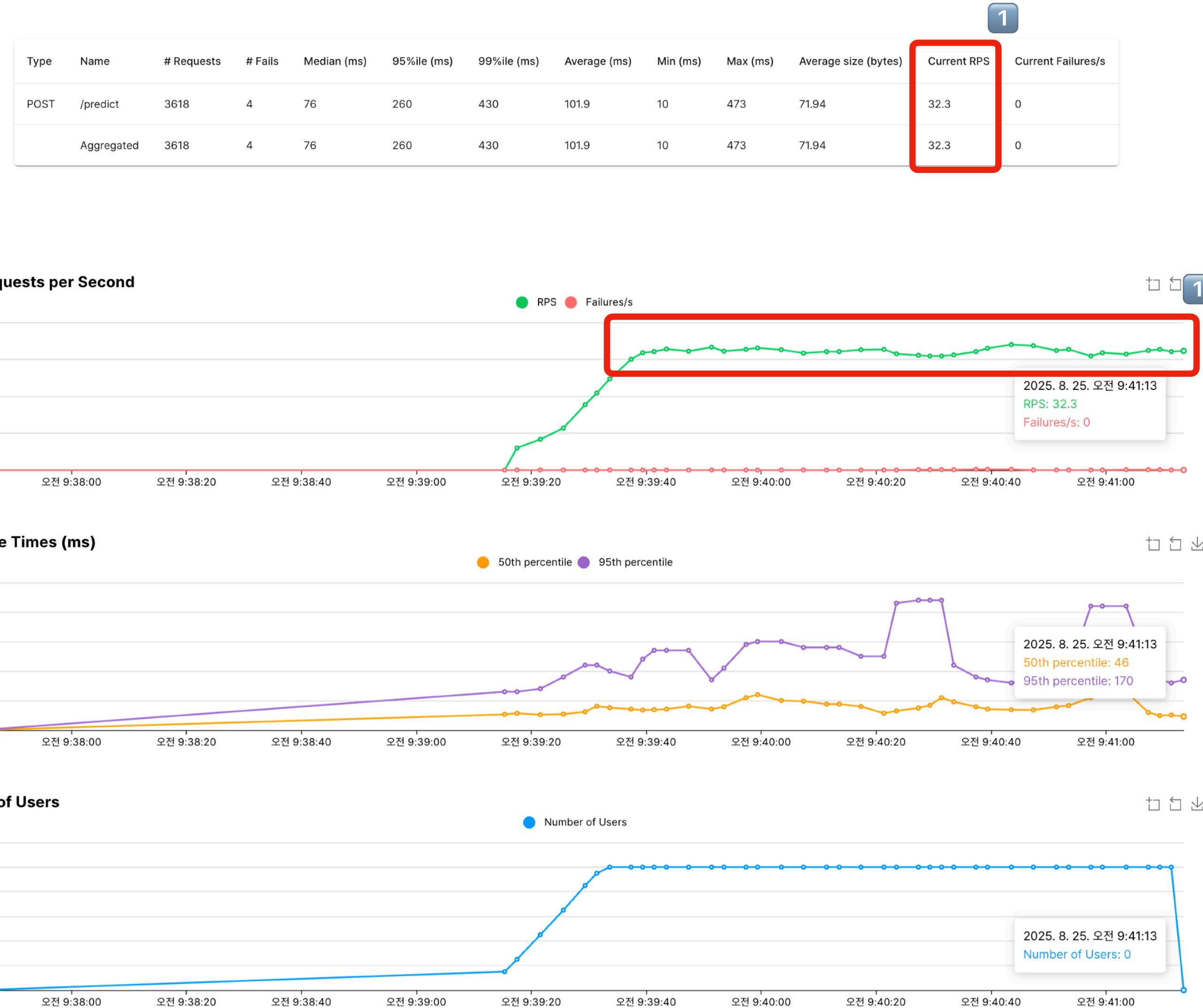
# 부하 테스트 결과 분석

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/predict	3618	4	76	260	430	101.9	10	473	71.94	32.3	0
	Aggregated	3618	4	76	260	430	101.9	10	473	71.94	32.3	0



## 4.1 모델 배포 (API 개발)

# 부하 테스트 결과 분석



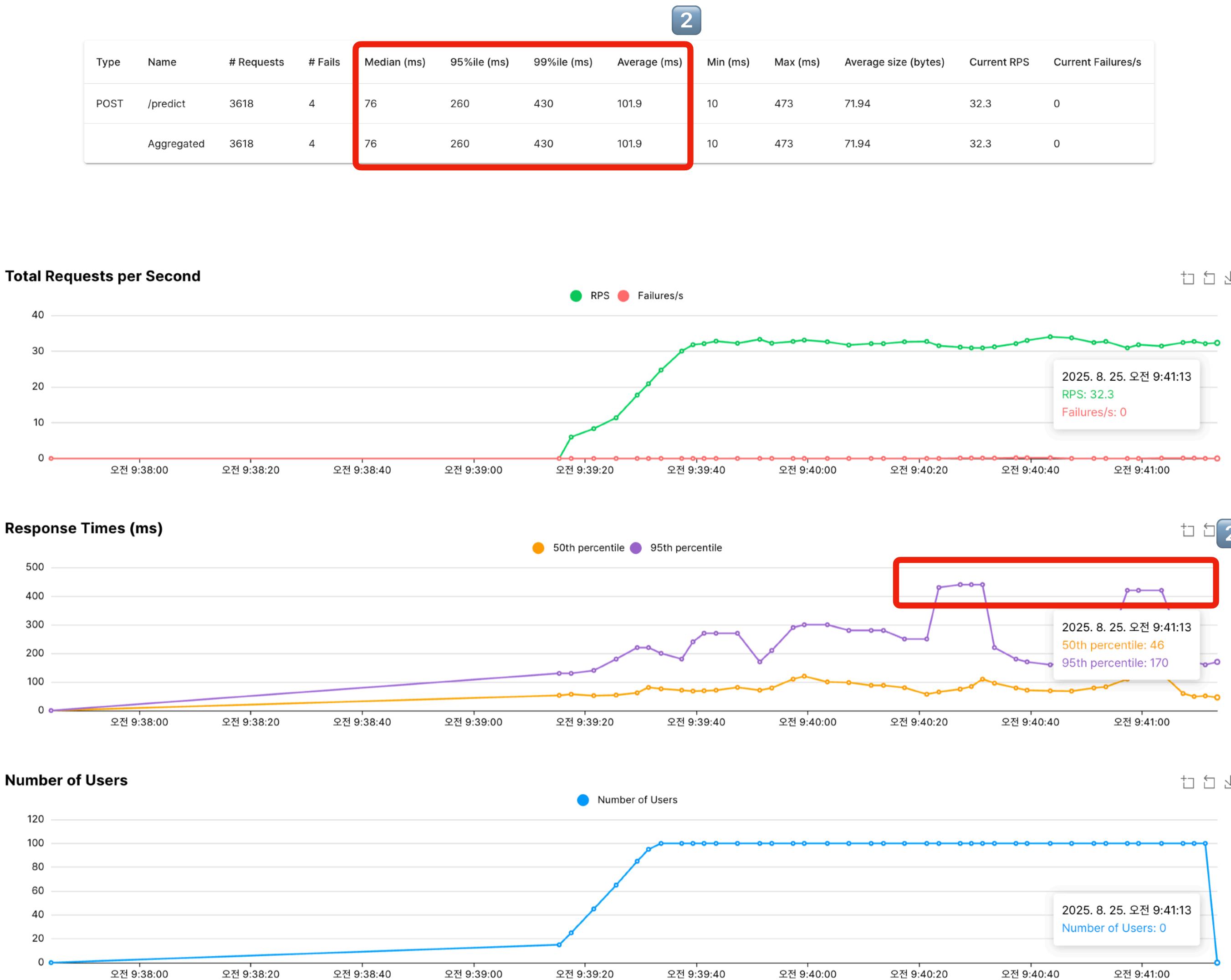
### ① 처리량 (Throughput)

- 사용자가 최대(100명)로 늘어난 후에도 **초당 요청 수(RPS, Requests per second)**가 약 32.3에서 **변화가 발생하지 않음**

→ 현재 상황에서 꾸준히 요청을 처리할 수 있음

## 4.1 모델 배포 (API 개발)

# 부하 테스트 결과 분석

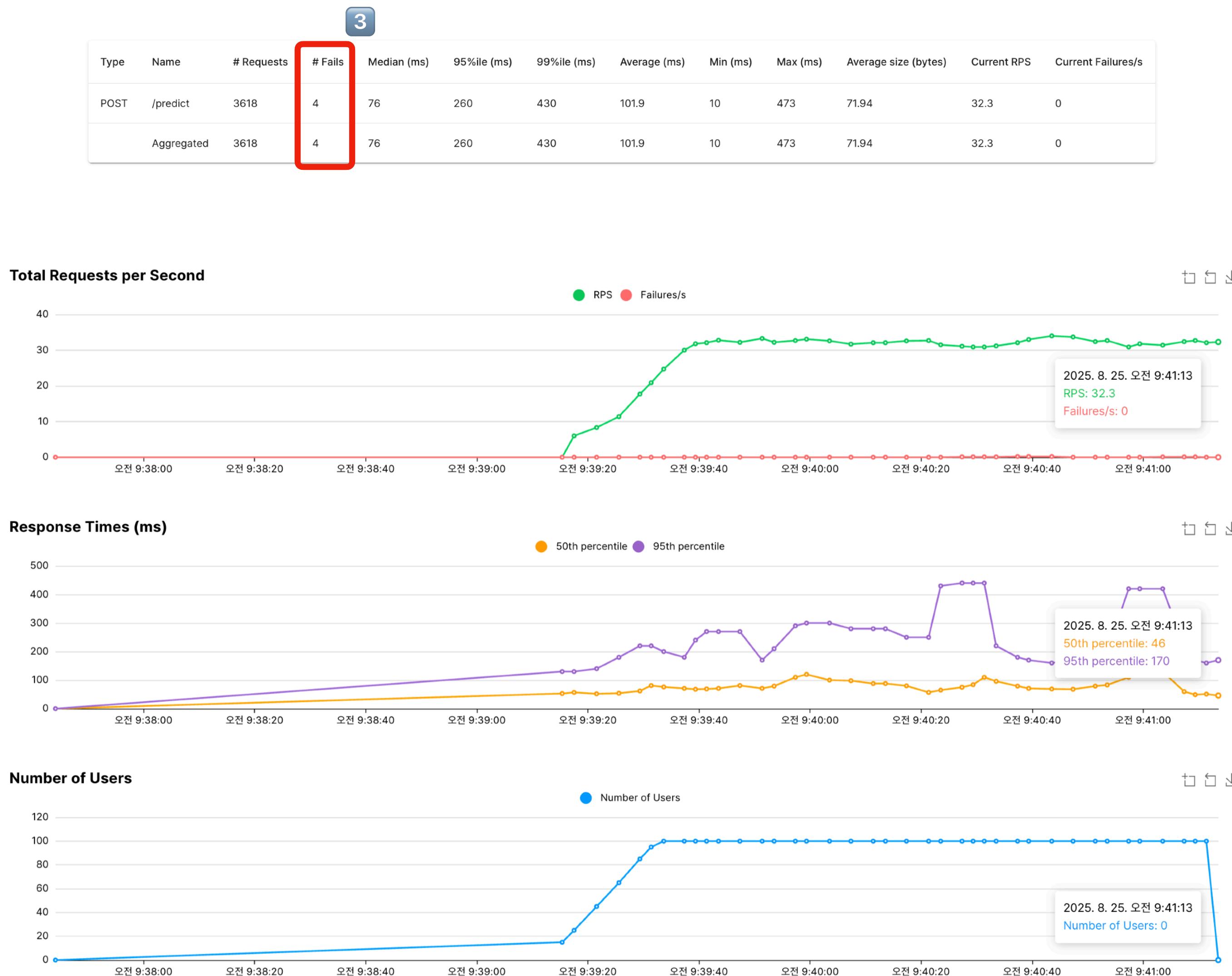


## 2 응답 시간 (Response Times)

- 중앙값(Median, 50th percentile)은 76ms로 안정적임  
(사용자의 50%는 빠른 응답속도로 사용)
- 95th percentile은 260ms로 사용자 수가 증가함에 따라 다소 불안정한 느낌이 있음
  - 특정 구간에선 400ms 이상으로 치솟음
- 중앙값과 95th percentile 사이의 차이에서 병목이 발생 할 수 있음을 확인 가능

## 4.1 모델 배포 (API 개발)

# 부하 테스트 결과 분석



## 3 안정성 및 에러 (Stability & Errors)

- 전체 3,618개 요청 중 4개 실패 → **실패율 0.11%**
  - 매우 낮은 실패율이지만 **원인 파악이 필요**
- RemoteDisconnected('Remote end closed connection without response')
  - 요청이 서버로 도착하기도 전에 거부되어 발생한 에러로

### BentoML 차원에서 확인할 수는 없음

- 서비스를 CPU 2개로만 띄워서 발생할 확률이 높음
- Locust로 부하 테스트 시 부하를 더 점진적으로 증가시키는 것도 방법

## 4.1 모델 배포 (API 개발)

# Docker 설정 파일 개발

## 1 Dockerfile

```
FROM python:3.12-slim-bookworm
LABEL maintainer="otzslayer@gmail.com"
COPY --from=ghcr.io/astral-sh/uv:latest /uv /uvx /bin/

ARG USER_HOME=/home/codespace
ARG UTIL_PATH=utils
ARG API_PATH=api

RUN groupadd --gid 1000 codespace \
    && useradd --uid 1000 --gid codespace --shell /bin/bash --create-home codespace

COPY --chown=codespace:codespace ${UTIL_PATH}/ ${USER_HOME}/utils
COPY --chown=codespace:codespace ${API_PATH}/bentofile.yaml ${USER_HOME}/
COPY --chown=codespace:codespace ${API_PATH}/requirements.txt ${USER_HOME}/
COPY --chown=codespace:codespace ${API_PATH}/services.py ${USER_HOME}/
COPY --chown=codespace:codespace ${API_PATH}/src/ ${USER_HOME}/${API_PATH}/src

USER codespace
WORKDIR ${USER_HOME}/${API_PATH}
ENV PYTHONUNBUFFERED=1 \
    VIRTUAL_ENV="${USER_HOME}/${API_PATH}/.venv"

ENV PATH="${VIRTUAL_ENV}/bin:${USER_HOME}/.local/bin:${PATH}" ←————• 이 설정을 하지 않으면 컨테이너 내에서 BentoML의 경로를 찾지 못함

RUN uv init --python 3.12 \
    && uv venv --python 3.12 --seed \
    && uv pip install -r ${USER_HOME}/requirements.txt
```

## 4.1 모델 배포 (API 개발)

# Docker 설정 파일 개발

## 2 docker-compose.yml

api/docker/docker-compose.yml

```

services:
  bentoml_service:
    build:
      context: ../..
      dockerfile: api/docker/Dockerfile
    image: credit_score_classification-deploy:latest
    container_name: credit_score_classification_deploy
    volumes:
      - ${HOME}/airflow/artifacts:/home/codespace/artifacts
      - ${HOME}/bentoml:/home/codespace/bentoml
    environment:
      PYTHONPATH: /home/codespace
      ARTIFACTS_PATH: /home/codespace/artifacts
      FEATURE_STORE_URL: mysql+pymysql://root:root@mariadb:3306/mllops
    ports:
      - "3000:3000"
    command: >
      bentoml serve services:CreditScoreClassifier
  networks:
    mllops_network:
      name: mllops_network
      external: true

```

.drwxr-xr-x@ jayhan 14 Feb 11:10 .
.drwxr-xr-x@ jayhan 14 Feb 11:10 api
.drwxr-xr-x@ jayhan 14 Feb 11:10 src
.+ \_\_init\_\_.py
.+ db.py
.+ models.py
.+ schemas.py
artifacts
bentoml
utils
.+ \_\_init\_\_.py
.+ callbacks.py
.+ common.py
.+ dates.py
! bentofile.yaml
requirements.txt
services.py

#### 4.1 모델 배포 (API 개발)

## Docker로 API 서비스 실행

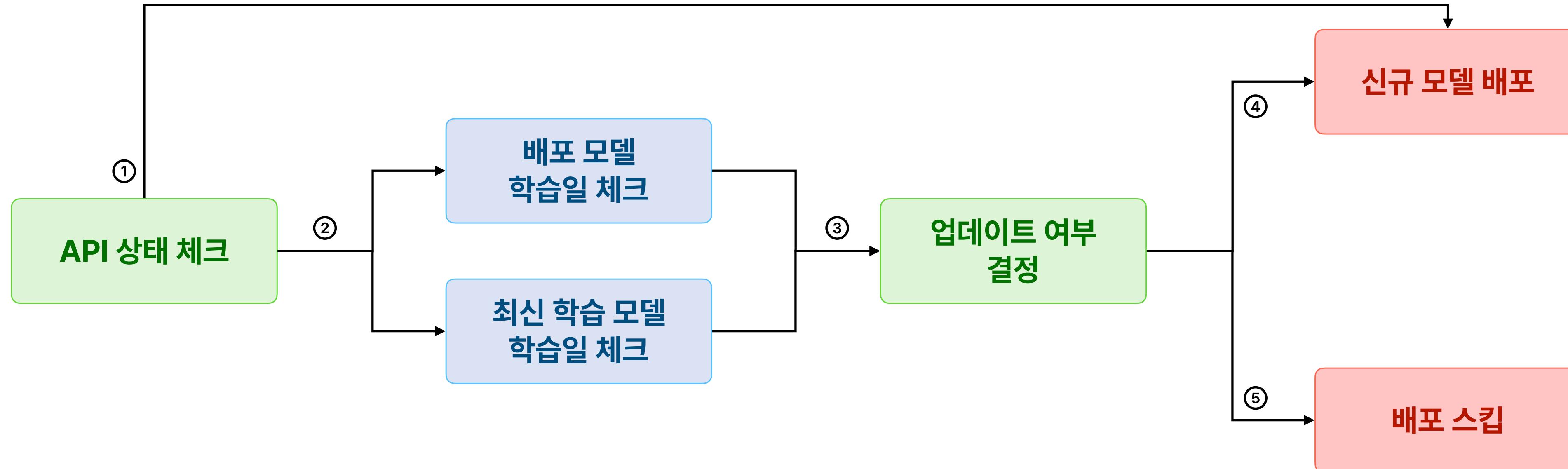
---

```
# api 폴더로 이동하여 Docker로 API 서비스 실행 (백그라운드 실행)
$ cd api
$ cd docker
$ docker compose up -d
```

## **4.2 지속적 배포 구현**

#### 4.2 지속적 배포 구현

## 지속적 배포 DAG



- ① API 상태를 체크하여 **API가 작동하지 않을 때는 바로 신규 모델 배포**
- ② API가 활성화되어 있는 상태인 경우 **현재 배포되어 있는 모델의 학습일과 가장 최근에 학습된 모델의 학습일을 체크**
- ③ 두 학습일을 다음 Task로 넘겨 비교
- ④ 배포된 모델보다 학습된 모델이 더 최근에 생성되었거나 배포된 모델이 없다면 최신 학습 모델로 모델 배포
- ⑤ 배포된 모델이 최신이거나 학습된 모델이 없는 경우 배포 스킵

#### 4.2 지속적 배포 구현

## 지속적 배포 DAG 개발

---

`pipelines/continuous_deployment/continuous_deployment_dag.py`

# TODO 부분 코드 작성

#### 4.2 지속적 배포 구현

## HTTP 응답 상태 코드

---

코드	의미	설명	예시
<b>200</b>	<b>OK</b>	요청이 성공적으로 처리됨 (일반적인 성공 응답)	데이터 조회 성공
<b>201</b>	<b>Created</b>	요청이 성공적으로 처리되었으며, 새로운 리소스가 생성됨 (POST 요청 결과)	데이터 생성 성공
<b>400</b>	<b>Bad Request</b>	클라이언트의 요청이 잘못됨 (유효하지 않은 데이터, 형식 오류 등)	잘못된 요청 (폼 입력 오류)
<b>401</b>	<b>Unauthorized</b>	인증이 필요함	인증 필요 (로그인 필요)
<b>403</b>	<b>Forbidden</b>	접근이 금지됨 (권한 부족)	권한 부족 (접근 거부)
<b>404</b>	<b>Not Found</b>	요청한 리소스를 찾을 수 없음	리소스 없음 (잘못된 URL)
<b>500</b>	<b>Internal Server Error</b>	서버 내부 오류 발생	서버 오류 발생
<b>502</b>	<b>Bad Gateway</b>	서버가 게이트웨이 역할을 하며 다른 서버로부터 잘못된 응답을 받음	백엔드 서버의 잘못된 응답
<b>503</b>	<b>Service Unavailable</b>	서버가 과부하 상태이거나 유지보수 중이라 사용할 수 없음	과부하나 자원 부족으로 인한 서버 응답 불가

#### 4.2 지속적 배포 구현

# PythonOperator vs BranchPythonOperator

## PythonOperator



- 일반적인 Python 함수를 실행하는 Operator
- DAG 내에서 특정 Python 함수를 실행하고 결과 반환



- 단순히 지정된 Python 함수를 실행
- 실행 결과를 사용할 수도 있고 사용하지 않을 수도 있음
- **DAG의 흐름을 바꾸지 않음**

## BranchPythonOperator



- 특정 조건에 따라 DAG의 실행 흐름을 **분기(branching)**할 수 있는 Operator
- Python 함수의 반환 값이 실행할 다음 Task를 결정함



- DAG의 실행 흐름을 동적으로 변경할 수 있음
- **하나 이상의 Task ID를 반환해야 함**
- 선택된 Task만 실행되며, 선택되지 않은 Task는 Skipped 상태가 됨

## 4.2 지속적 배포 구현

# DAG 실행 결과 확인

**Dag**  
credit\_score\_classification\_cd

**Home** **Dags** **Assets** **Browse** **Admin**

**Options** **Trigger** **Reparse Dag** **Latest Dag Version**

**credit\_score\_classification\_cd** **Schedule** **Latest Run** **Next Run** **Owner** **Tags**

2025-08-25, 17:08:42 **✓** user lgcns, mlops v1

**Overview** **Runs** **Tasks** **Events** **Code** **Details**

Last 24 hours 2025-08-24, 17:08:05 - 2025-08-25, 17:08:05

**0 Failed Tasks** **0 Failed Runs**

**Last 2 Dag Runs**

Run After	Duration (seconds)
2025-08-25, 05:08:07	19.76
2025-08-25, 05:08:42	0.59

**Task Details**

get\_branch\_by\_api\_status  
get\_deployed\_model\_creation\_time **skipped**  
get\_latest\_trained\_model\_creation...  
decide\_update  
deploy\_new\_model  
skip\_deployment

**Logs** **Rendered Templates** **XCom** **Audit Logs** **Code** **Details** **Asset Events**

All Log Levels

0 Task was skipped, no logs available.

# **Wrap-up**

## Wrap-up

# Further Readings

---

## Books

- [\*\*Practical MLOps - Noah Gift\*\*](#): AWS, GCP, Azure CSP 3사 환경 기반의 MLOps 구축 가이드
- [\*\*머신러닝 시스템 설계 - Chip Huyen\*\*](#): ML 시스템 개발의 실전 적용
- [\*\*MLOps 구축 가이드북 - 김남기\*\*](#): Level 0 MLOps부터 Level 1 MLOps까지, 그리고 단위테스트를 포함한 MLOps 가이드



## Blogs & Articles

- [\*\*Google Cloud MLOps Guide\*\*](#)
- [\*\*Made With ML\*\*](#): MLOps 전단계 튜토리얼
- [\*\*Eugene Yan's Blog\*\*](#)
- [\*\*Chip Huyen's Blog\*\*](#)

**고생하셨습니다 !**