

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 hazy, light-colored mist.

2025

# MLOps 심화

들어가며

# 시작하기에 앞서

---

## 본 강의의 핵심

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

## 핸즈온 실습

- ▶ 실습 위주로 실제 데이터를 전처리하고 모델을 학습, 배포하는 작업을 자동화하는 연습
- ▶ 각 코드의 중요 부분을 비워놓고 수강생이 해당 부분을 채우는 방식
- ▶ 되도록 쉘 환경을 사용

## 파일럿 강의입니다!

- ▶ 난이도 이슈
- ▶ 시간 배분
- ▶ 여러분의 피드백

들어가며

# 강의 일정

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

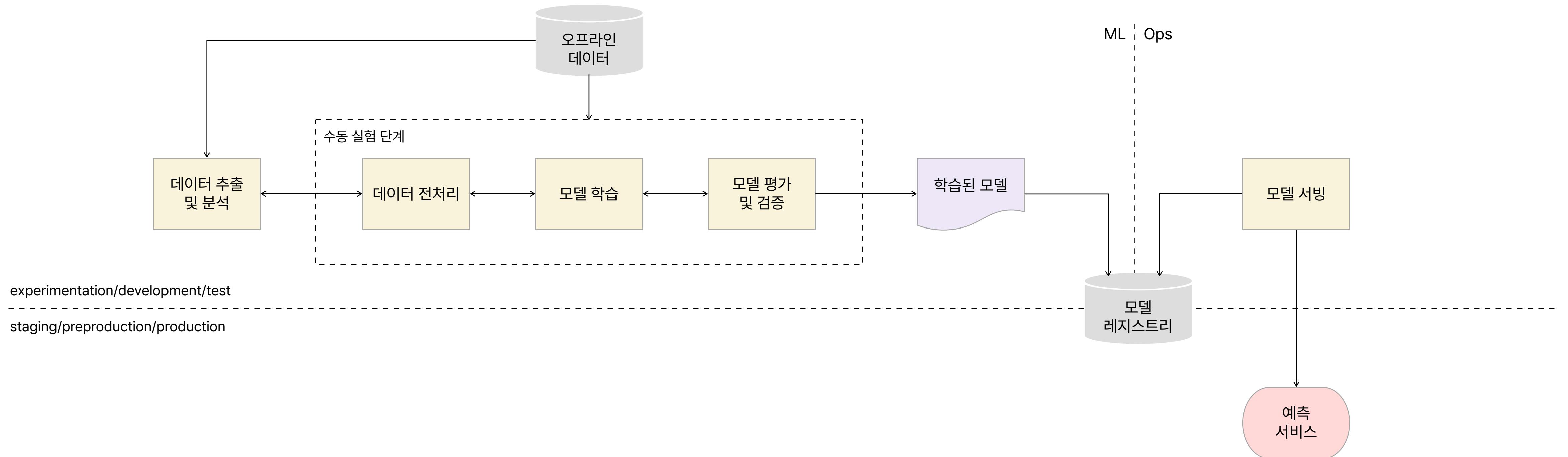
# **1.1 MLOps 프로세스 살펴보기**

## **— Google의 단계별 비교**

## 1.1 MLOps 프로세스 살펴보기

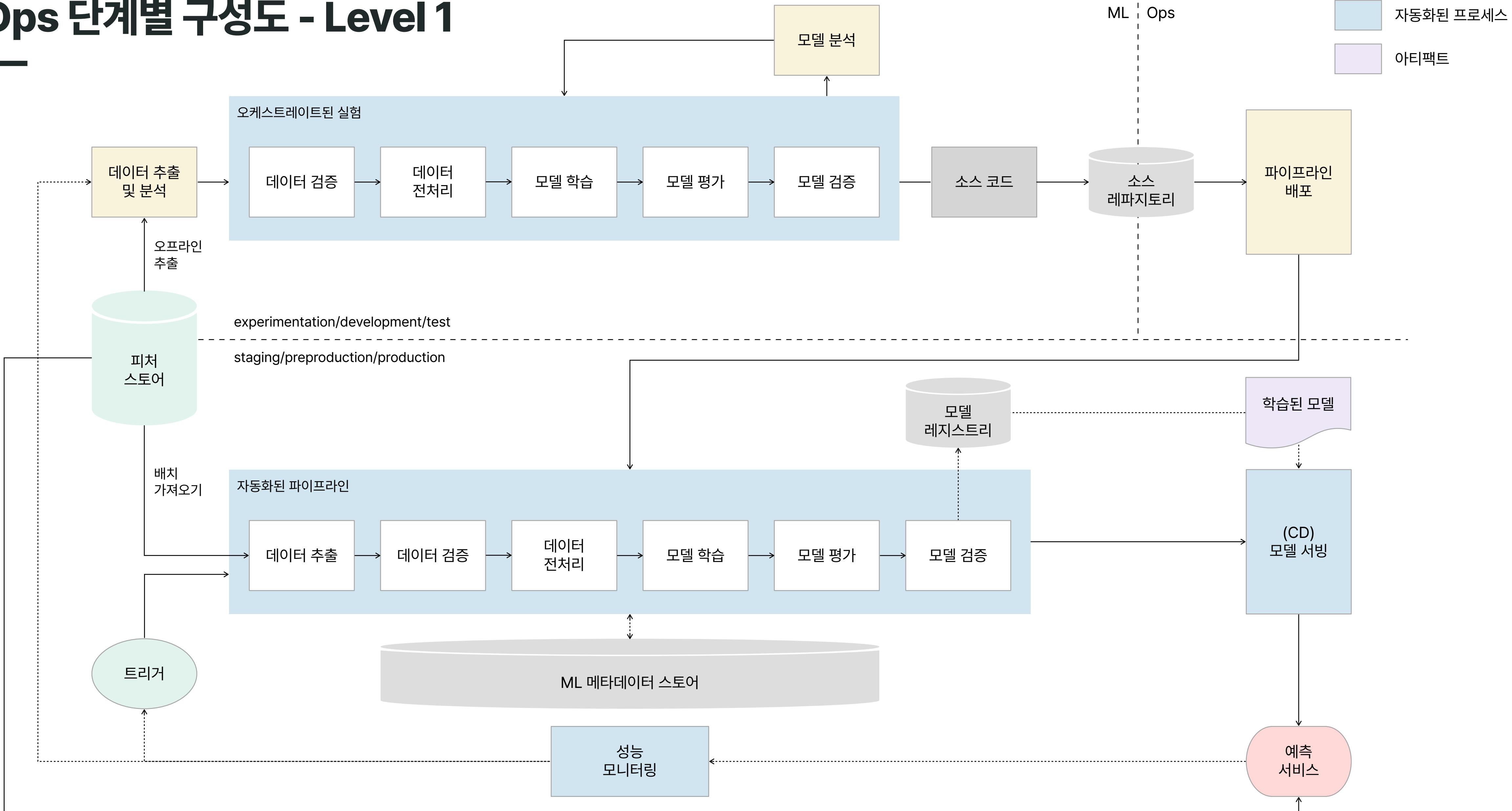
# MLOps 단계별 구성도 - Level 0

- 수동 프로세스
- 자동화된 프로세스
- 아티팩트



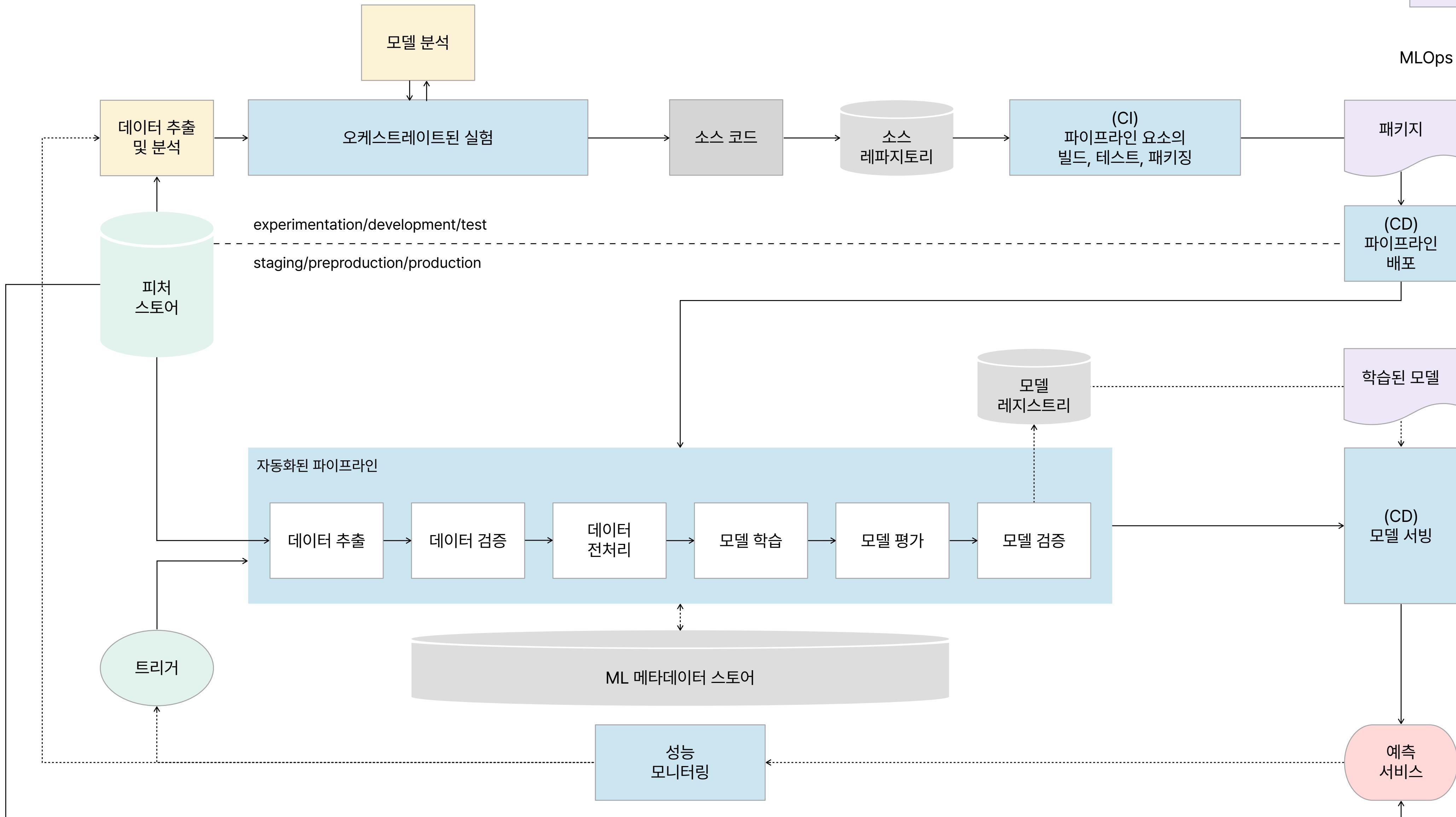
## 1.1 MLOps 프로세스 살펴보기

# MLOps 단계별 구성도 - Level 1



## 1.1 MLOps 프로세스 살펴보기

# MLOps 단계별 구성도 - Level 2

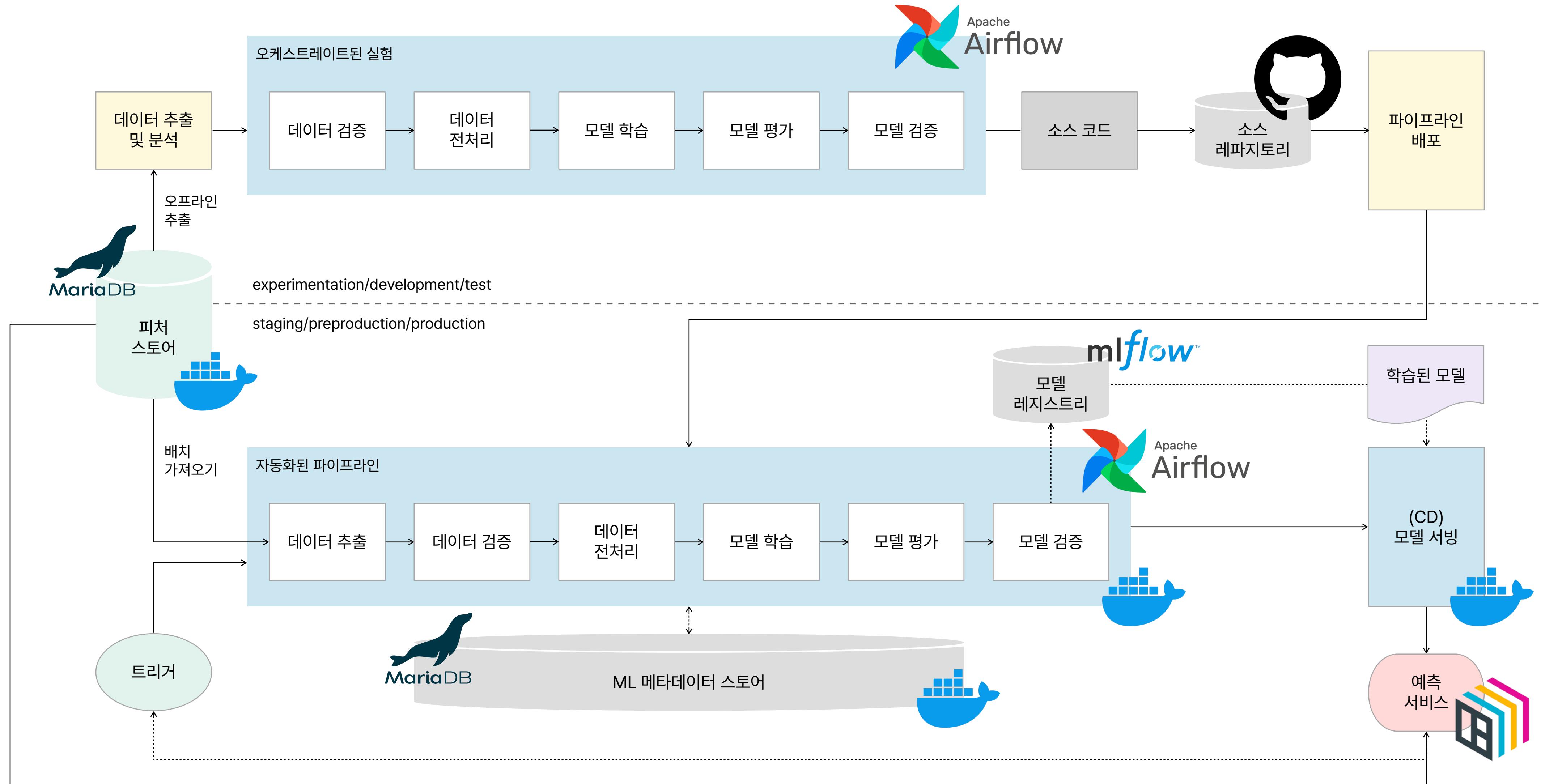


## **1.2 활용 기술 스택 알아보기**

## 1.2 활용 기술 스택 알아보기

# 실습 워크플로우

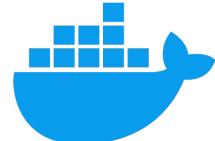
- 수동 프로세스
- 자동화된 프로세스
- 아티팩트



## 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'. The repository name is 'advanced-mlops'. The 'Code' tab is selected. The repository is public and has 1 branch ('main') and 0 tags. The commit history shows the first commit by 'otzslayer' at 099412c, 2 hours ago, which includes changes to 'api', 'db', 'pipelines', 'utils', '.env', '.gitignore', and 'README.md'. The 'About' section indicates no description, website, or topics are provided. The 'Fork' button is highlighted with a red box.

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	Change	Time
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

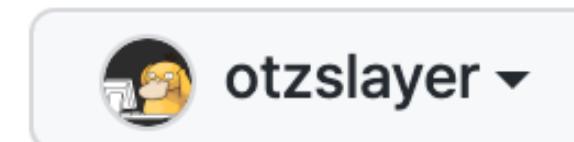
## 1.3 개발환경 설정

# 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

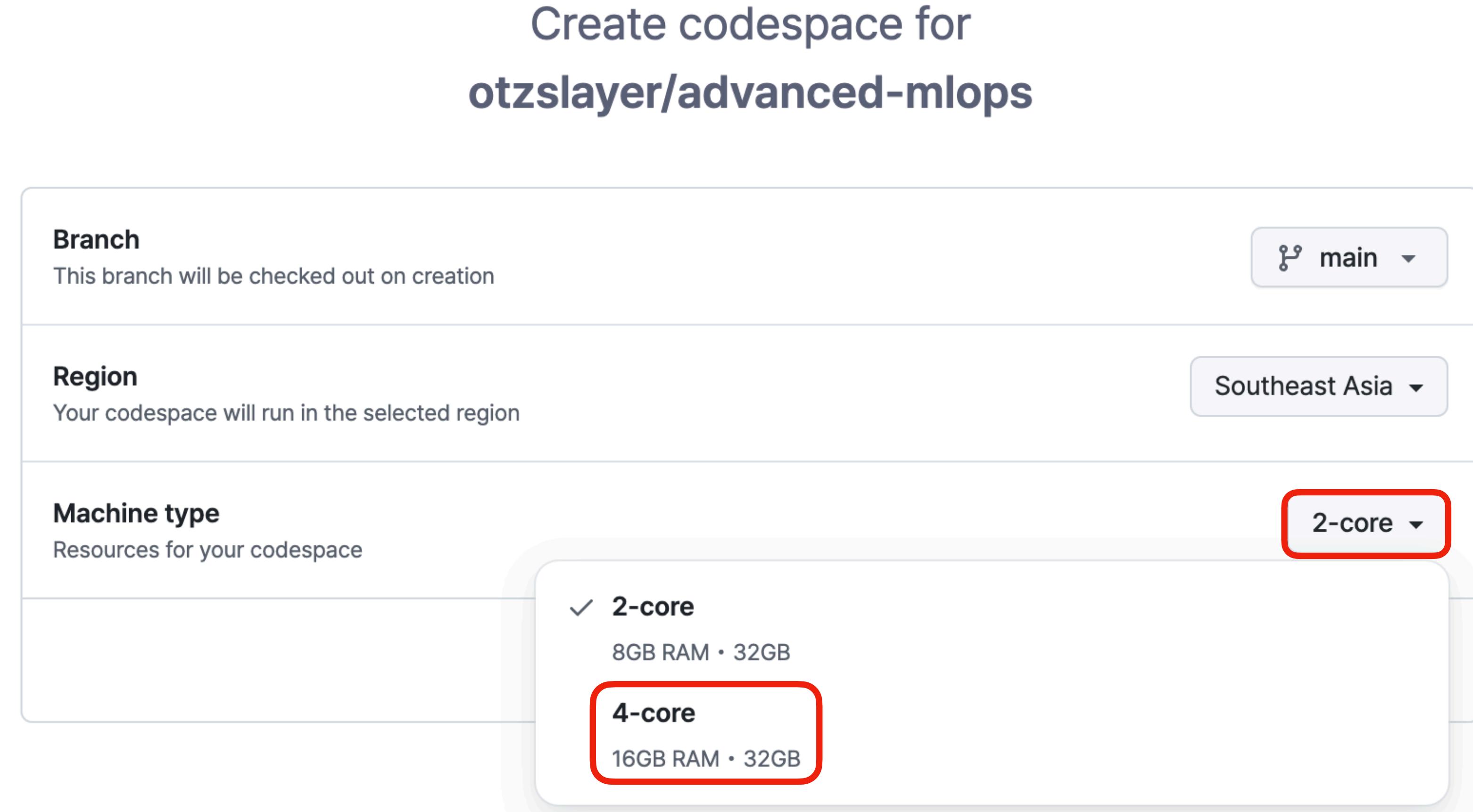
The screenshot shows a GitHub repository page for 'otzslayer / advanced-mlops'. The 'Code' tab is selected. In the top right, there are buttons for 'Pin', 'Unwatch 1', and 'Fork 0'. Below the navigation bar, it shows 'main' branch, '1 Branch', and '0 Tags'. A search bar says 'Go to file' with a 't' icon. To the right of the search bar is a 'Code' dropdown menu with a red border around it. The 'Codespaces' tab is selected in this menu. A sub-menu for 'Codespaces' is open, containing the following items:

- + (with a red circle around it)
- New with options... (highlighted with a red box)
- Configure dev container
- Set up prebuilds
- Manage codespaces
- Share a deep link
- What are codespaces?

Below the 'Codespaces' section, it says 'No codespaces' and 'You don't have any codespaces with this repository checked out'. There is a green 'Create codespace on main' button. At the bottom of the page, it says 'Codespace usage for this repository is paid for by otzslayer.' and 'No packages published'.

## 1.3 개발환경 설정

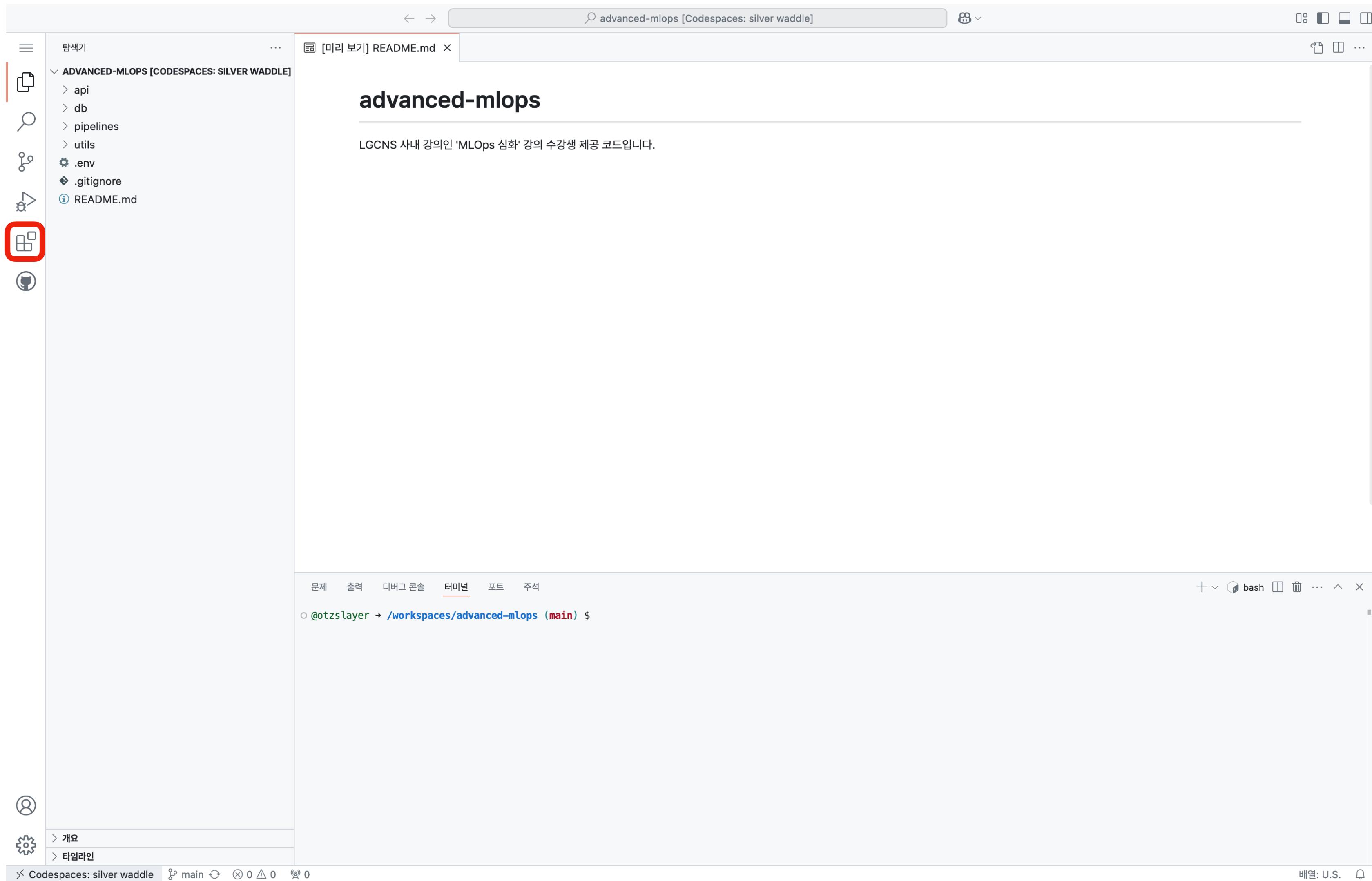
# Codespaces 생성



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

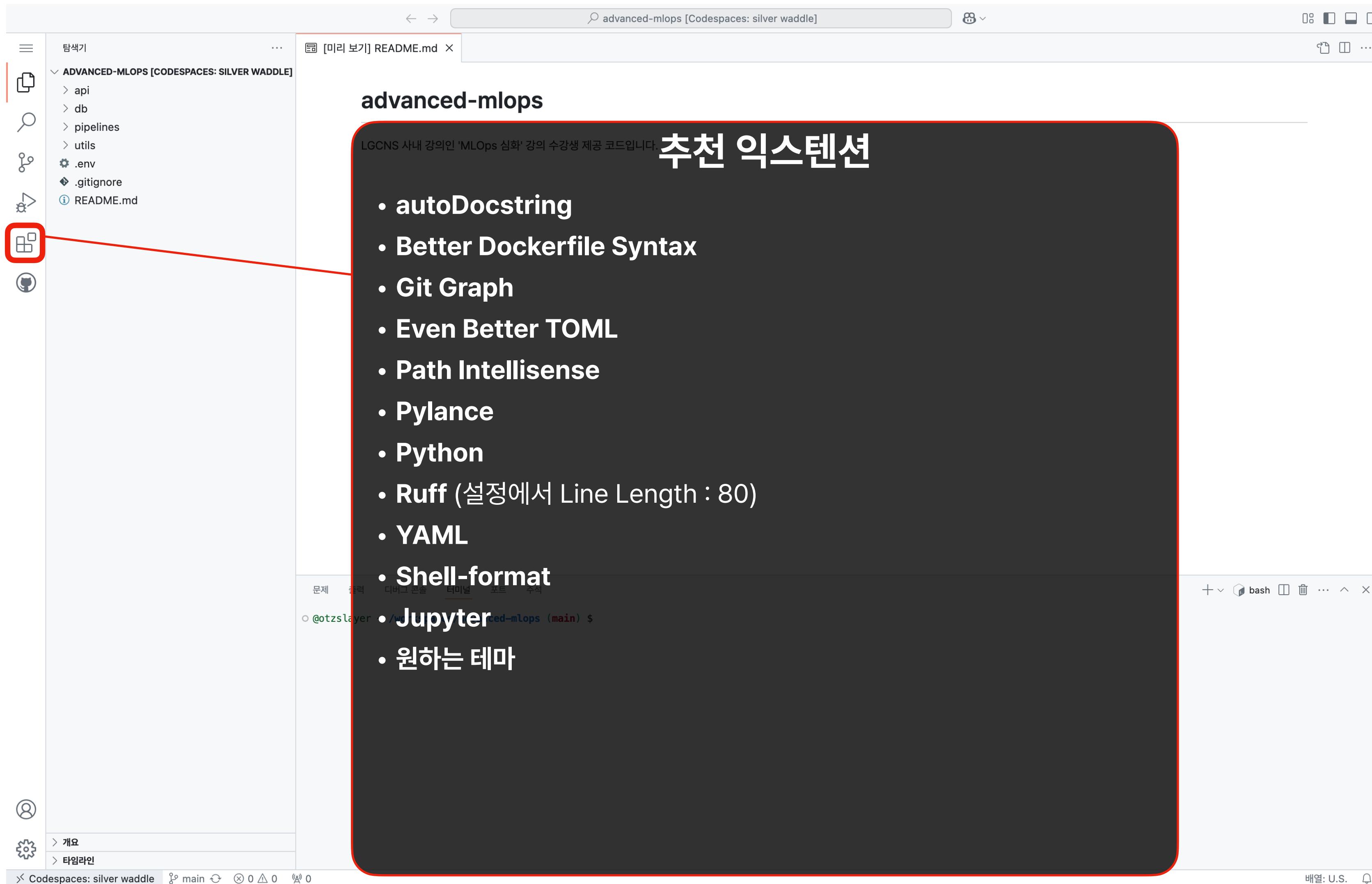
### 1.3 개발환경 설정

## Codespaces 생성



## 1.3 개발환경 설정

# Codespaces 생성



### 1.3 개발환경 설정

## 가상환경 설정

---

```
# 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
```

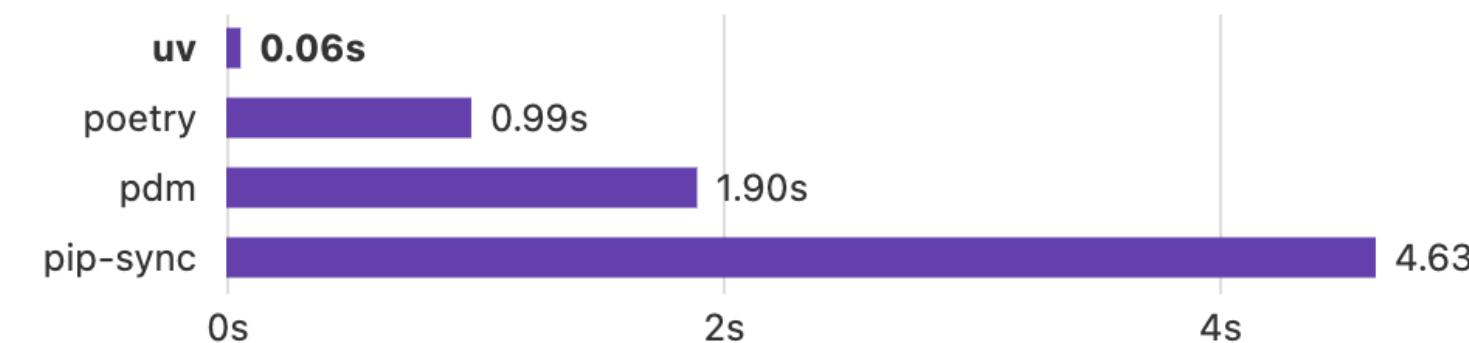
# 가상환경 설정

---

## 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 3.11.6 설치 포함)  
$ uv init --python 3.11.6  
$ uv venv --python 3.11.6  
$ 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 TERMCAPDIRS=/etc/termcap:/lib/termcap:/usr/share/termcap
```

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

```
# 불필요한 main.py 파일 삭제  
$ rm main.py
```

### 1.3 개발환경 설정

## 가상환경 설정

---

```
# 사용할 라이브러리 설치
$ uv add apache-airflow==2.10.4 apache-airflow-providers-mysql==5.7.4 mysqlclient==2.2.7 numpy==1.26.4 pandas==2.0.3 pymysql==1.1.0 scikit-learn==1.6.1 python-dotenv==1.0.1 mlflow==2.20.1 bentoml==1.3.21 catboost==1.2.7 pip==25.0.1 tqdm==4.67.1

# 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 = "≥3.11.6"
dependencies = [
    "apache-airflow=2.10.4",
    "apache-airflow-providers-mysql=5.7.4",
    "mysqlclient=2.2.7",
    "numpy=1.26.4",
    "pandas=2.0.3",
    "pymysql=1.1.0",
    "scikit-learn=1.6.1",
    "python-dotenv=1.0.1",
    "mlflow=2.20.1",
    "bentoml=1.3.21",
    "catboost=1.2.7",
    "pip≥25.0.1",
    "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
```



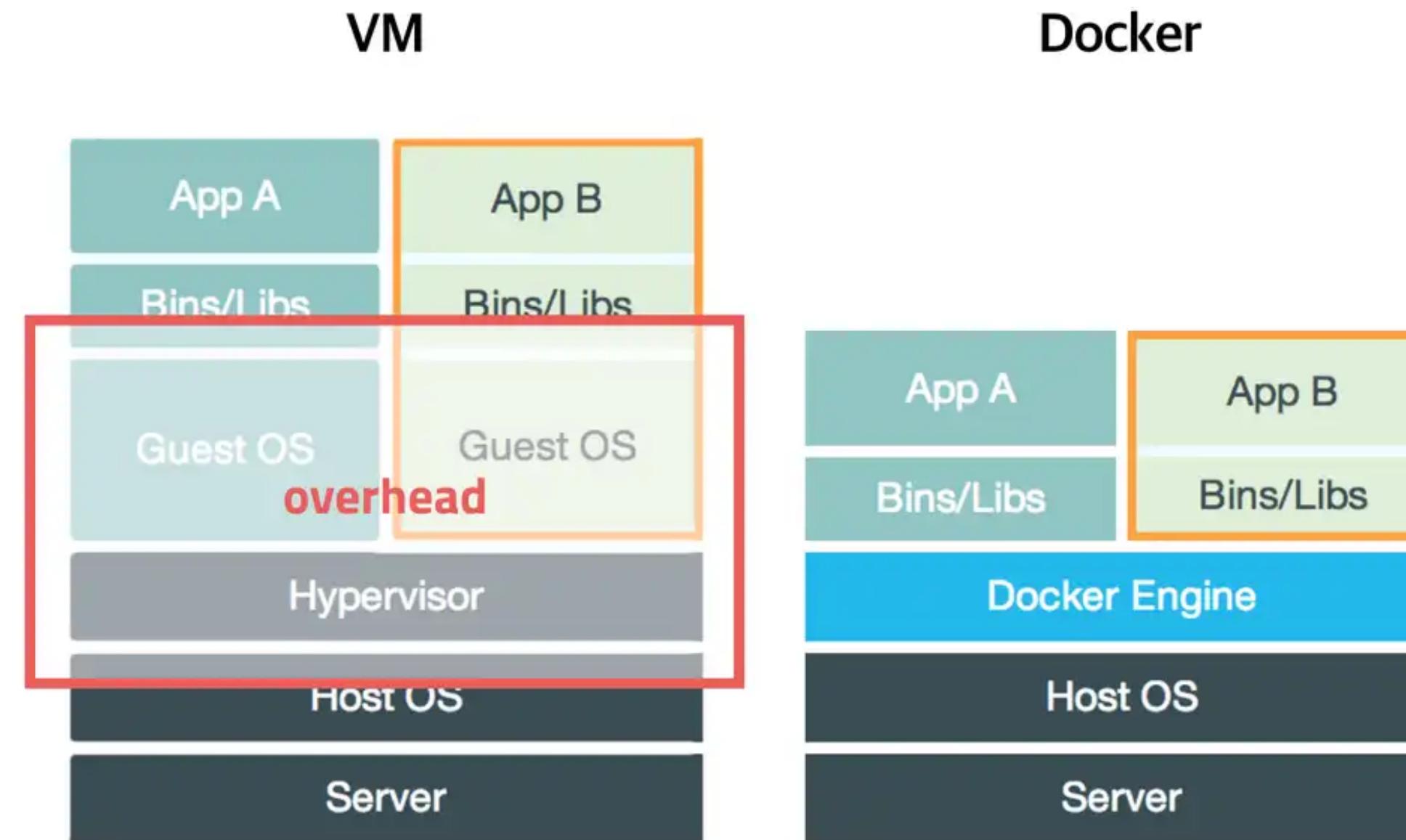
## M2. 활용 도구 익히기

1. Docker
2. SQLAlchemy
3. Airflow

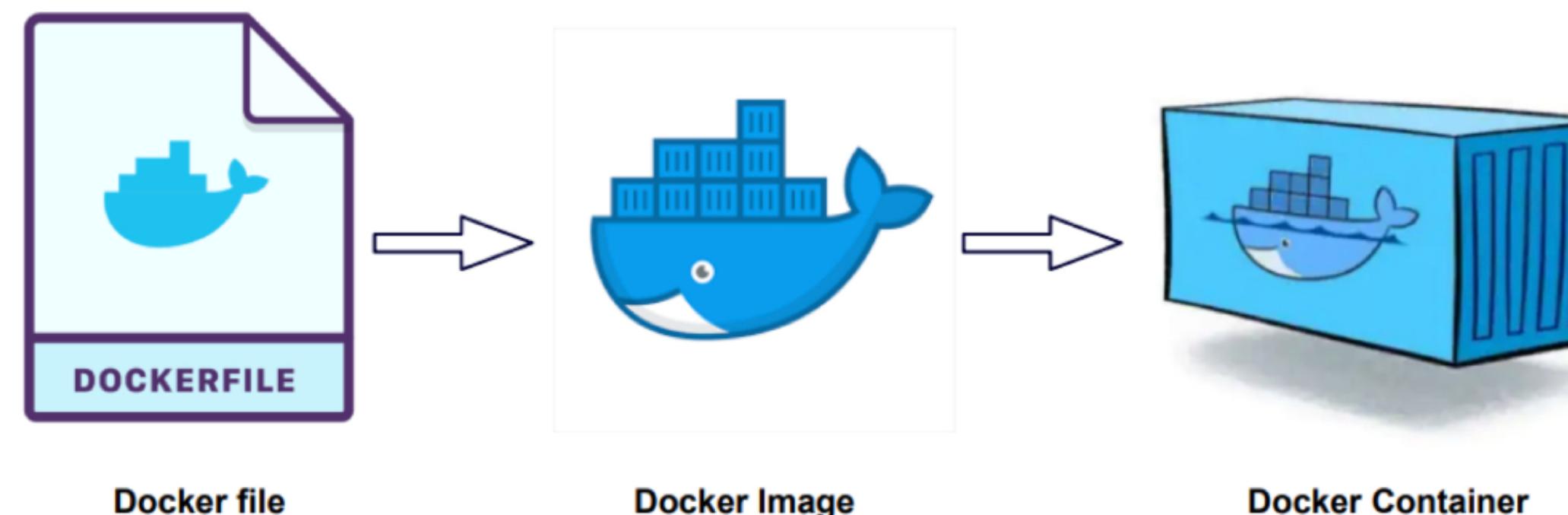
## **2.1 Docker**

## 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</b> 위해 사용
<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

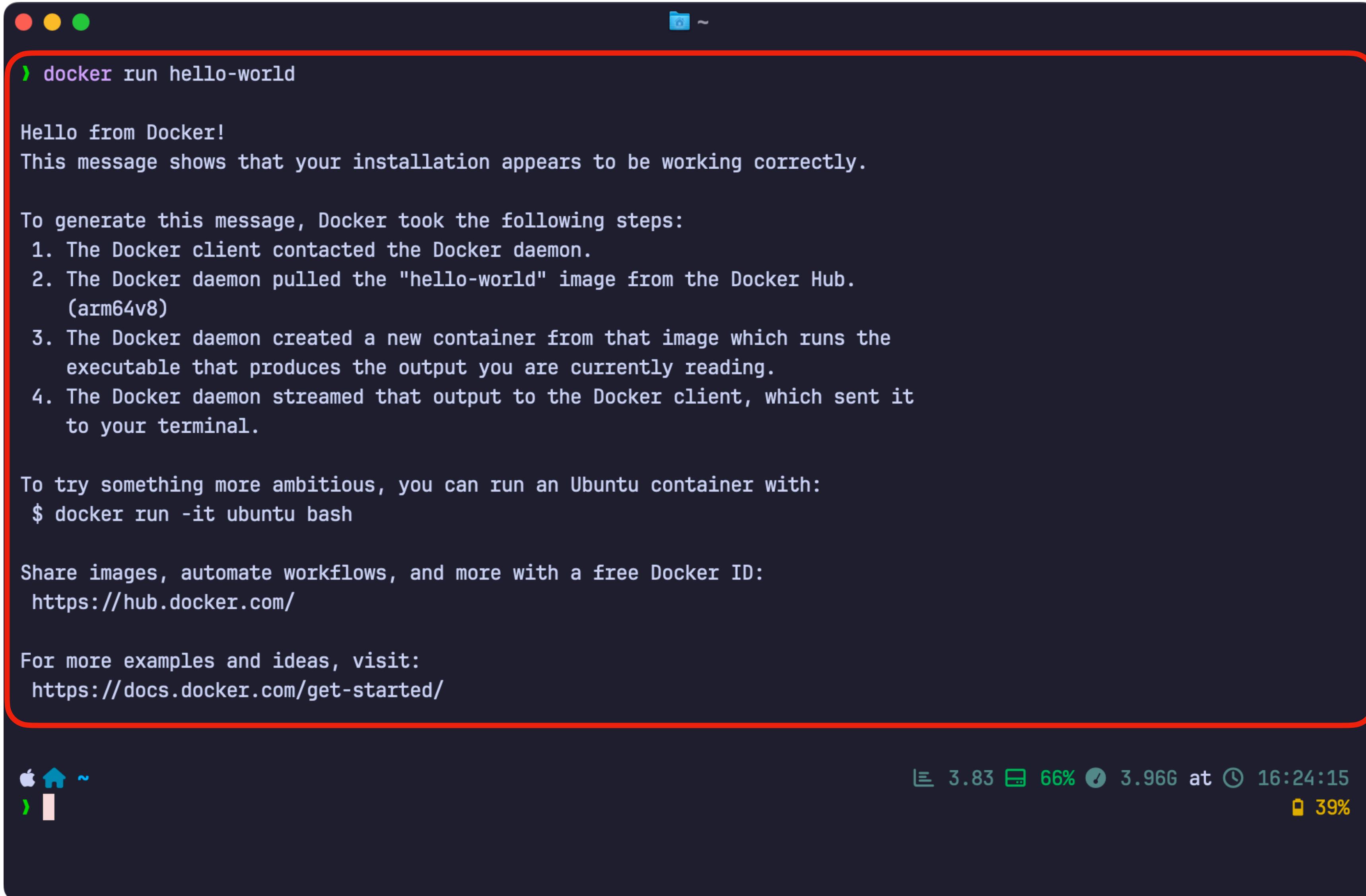
# Docker 기본 명령어

---

명령어	설명
<b>docker run</b>	컨테이너를 생성하고 실행합니다. (docker run [OPTIONS] IMAGE [COMMAND])
<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. (docker rm [CONTAINER_ID])
<b>docker rmi</b>	특정 Docker 이미지를 삭제합니다. (docker rmi [IMAGE_ID])
<b>docker exec</b>	실행 중인 컨테이너 내부에서 명령을 실행합니다. (docker exec -it [CONTAINER_ID] /bin/sh)

## 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

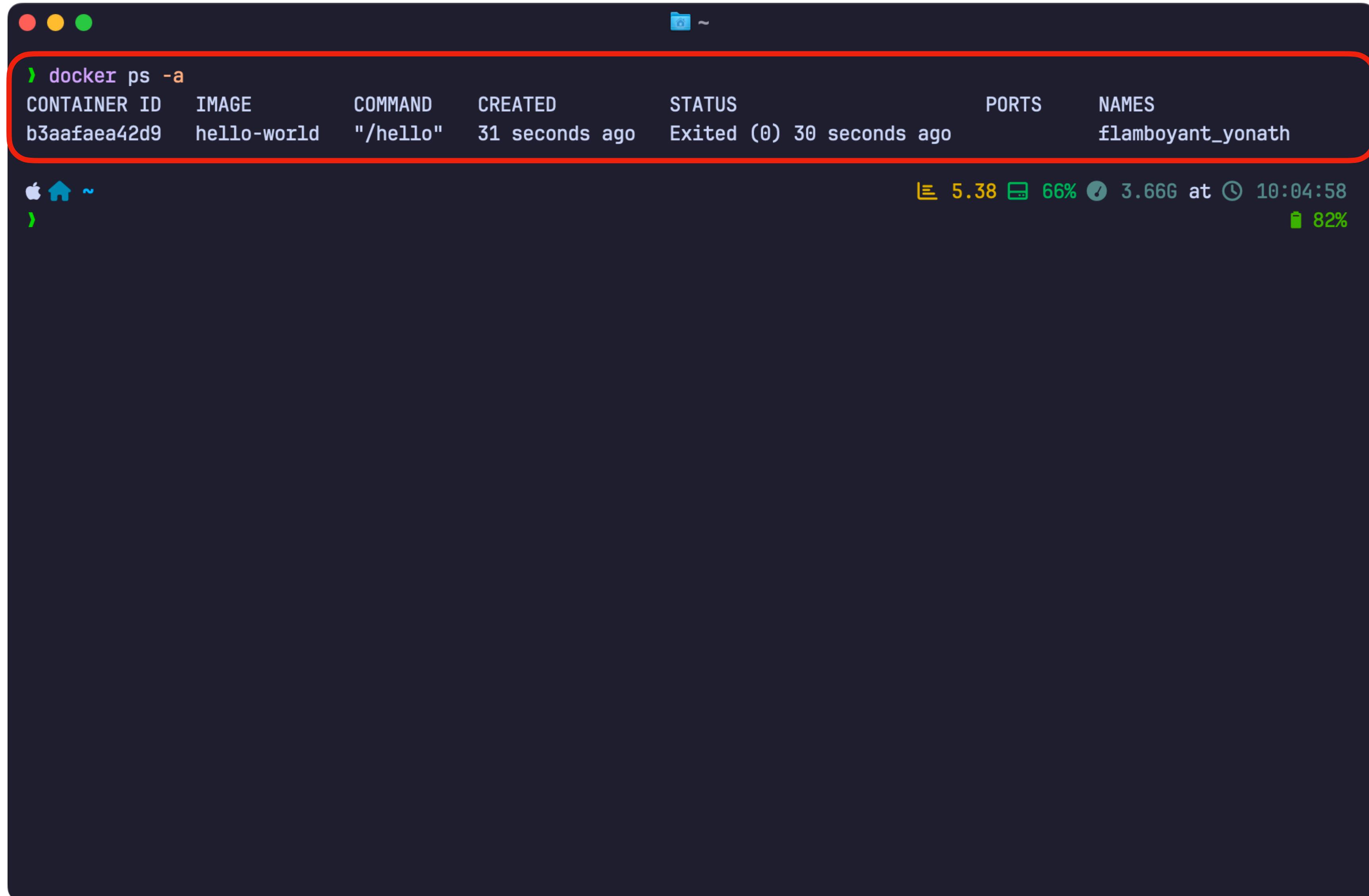
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

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 (39%), signal strength, and system time (16:24:15).

## 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
```

apple ~

5.38 66% 3.66G at 10:04:58  
82%

## 2.1 Docker

# Docker 기본 명령어

The screenshot shows a macOS terminal window with a dark theme. At the top, there's a file icon and a tilde (~) indicating the user's home directory. The terminal output is as follows:

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

At the bottom of the terminal, there's a system status bar showing:

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

## 2.1 Docker

# Docker 기본 명령어

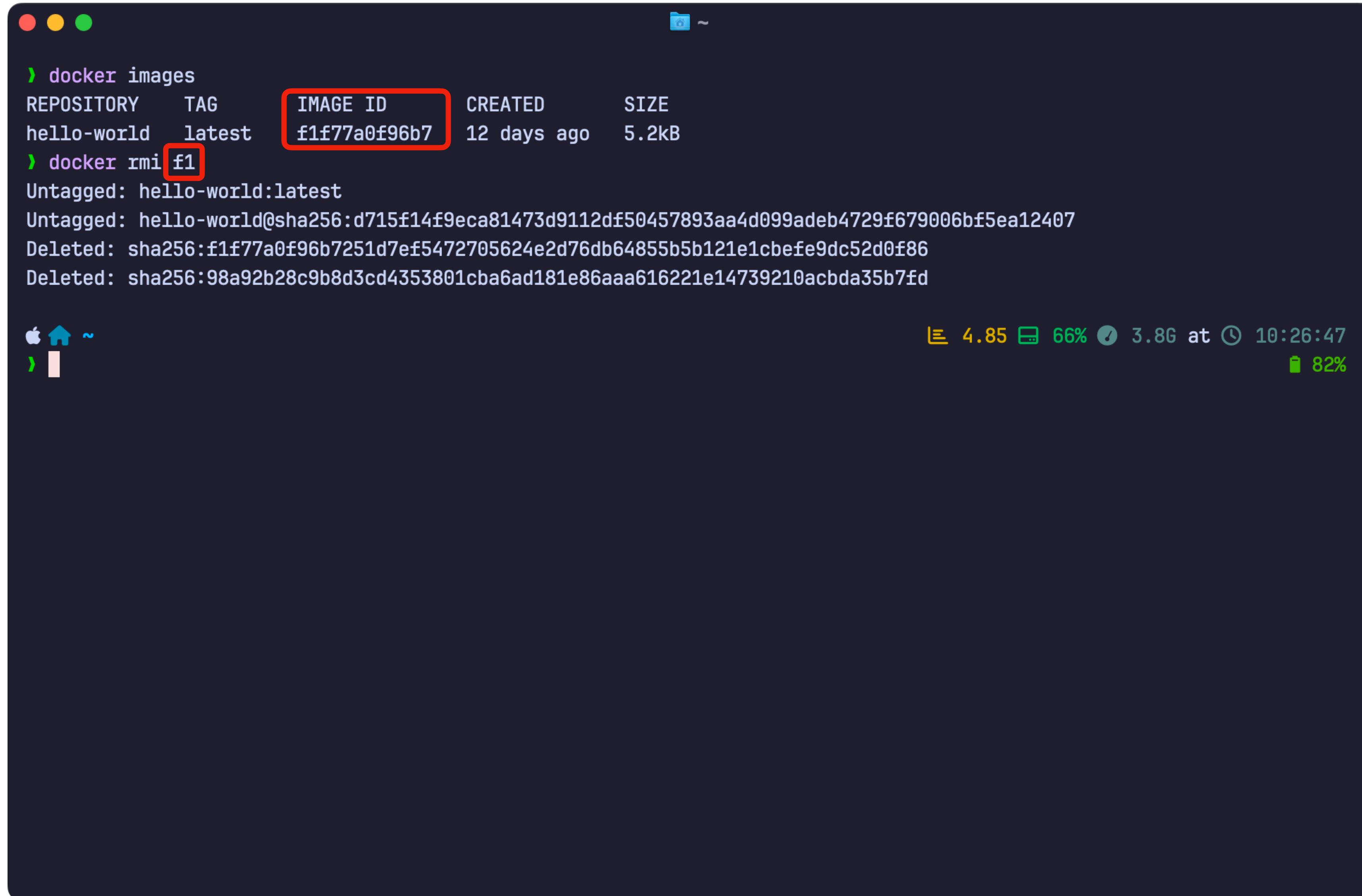
```
❯ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
hello-world    latest    f1f77a0f96b7  12 days ago  5.2kB
```

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 hash.

At the bottom right, system status icons show battery level at 82%, CPU usage at 4.85%, memory usage at 66%, and disk space at 3.8G.

```
> 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

4.85 66% 3.8G at 10:26:47
82%
```

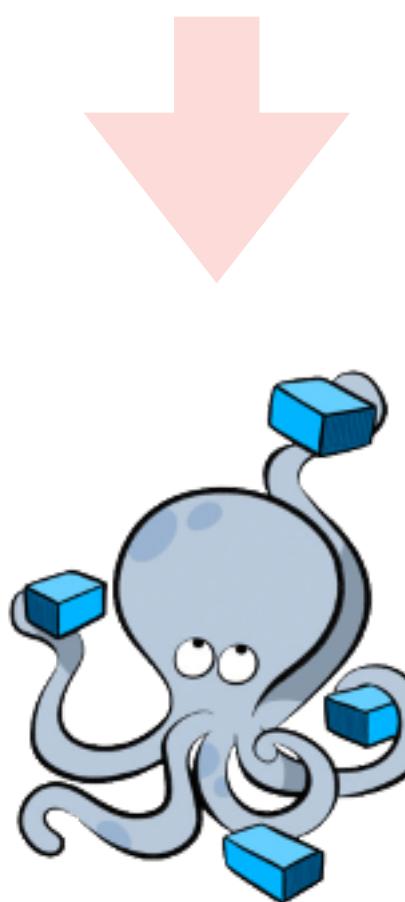
## 2.1 Docker

# 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

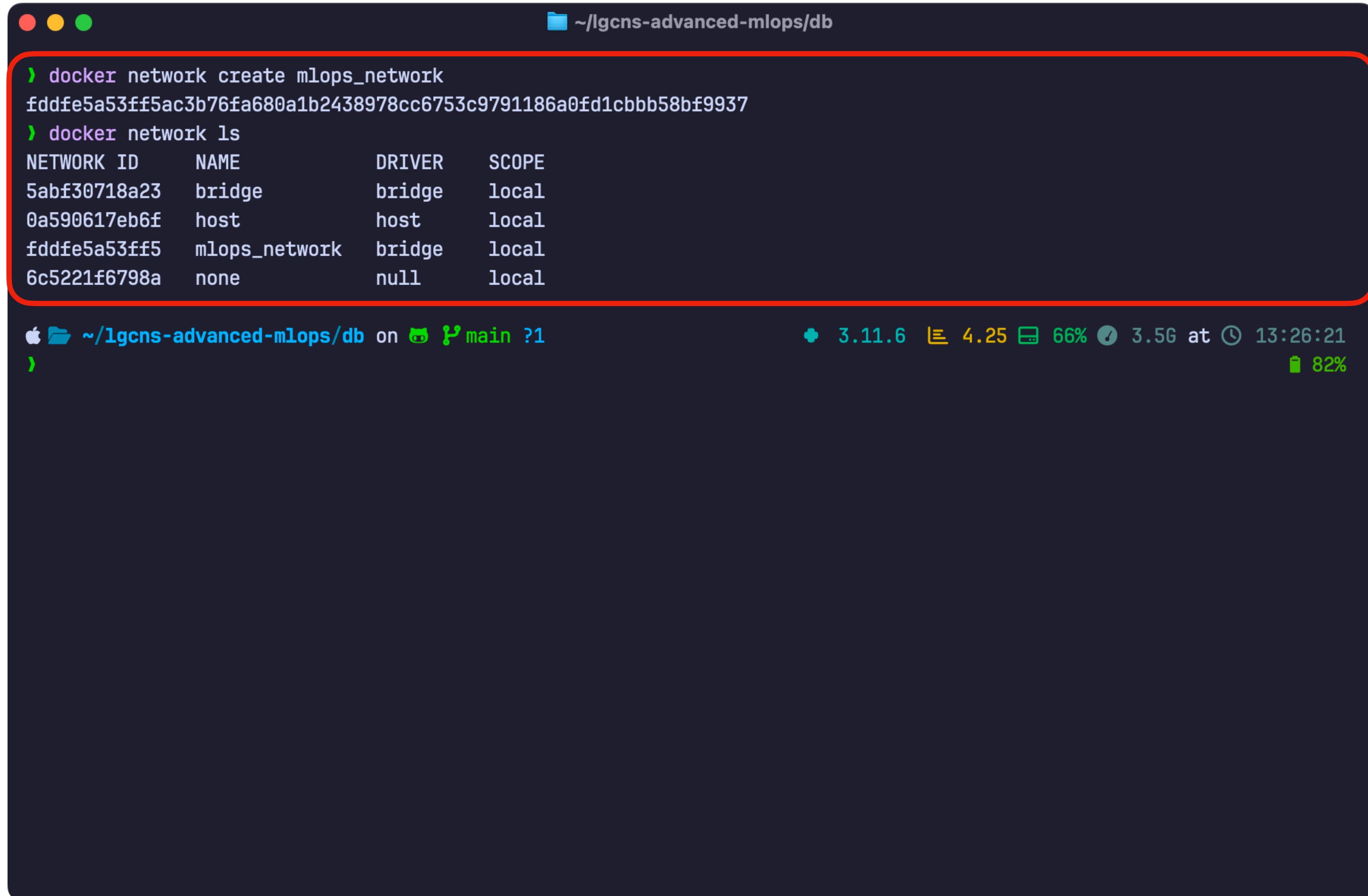
```

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



A screenshot of a macOS terminal window titled '~/.lgcns-advanced-mlops/db'. The window contains the following text:

```
> 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 terminal prompt shows the user is in a directory (~/.lgcns-advanced-mlops/db) on a main branch with a commit hash of ?1. The bottom right corner of the terminal shows system status: 3.11.6, 4.25, 66%, 3.5G at 13:26:21, and 82% battery.

## 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

```
● ● ●
~/lgcns-advanced-mlops/db

❯ docker compose up -d
[+] Running 20/20
  ✓ db Pulled
    ✓ d4ba87bb7858 Pull complete          27.3s
    ✓ 932d0da43793 Pull complete          10.8s
    ✓ dc35c765a732 Pull complete          10.8s
    ✓ ed88bae007bd Pull complete          10.9s
    ✓ 7734d2f7be98 Pull complete          11.0s
    ✓ e24600bac48c Pull complete          11.0s
    ✓ da05700210f6 Pull complete          11.8s
    ✓ ce763ae3d194 Pull complete          11.8s
    ✓ 1d496894956c Pull complete          11.8s
    ✓ c5ff3d92cd0a Pull complete          24.6s
    ✓ b65305c7f16f Pull complete          24.6s
  ✓ adminer Pulled
    ✓ 2c9750102c61 Pull complete          24.6s
    ✓ 1eb6bc35ca96 Pull complete          15.6s
    ✓ baf5a77f4b47 Pull complete          11.7s
    ✓ de9e0e1297b2 Pull complete          12.9s
    ✓ bd2fd75f19c8 Pull complete          12.9s
    ✓ 8fd82873adc1 Pull complete          12.9s
    ✓ 61f74b8ed8bd Pull complete          13.0s
  [+]
  [+]
  ✓ 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 code editor interface with a sidebar and a main workspace.

**File Explorer:** Shows a project structure under 'ADVANCED-MLOPS [CODESPACES: SILVER WADDLE]'. The 'docker-compose.yml' file is selected.

**Main Area:** Displays the content of the 'docker-compose.yml' file:

```

version: '3.8'
services:
  db:
    image: mariadb:10.8.2-rc-focal
    container_name: mariadb
    hostname: mariadb
    volumes:
      - ${PWD}/data:/tmp
      - ${PWD}/docker-entrypoint-initdb.d:/docker-entrypoint-initdb.d
    restart: always
    ports:
      - "3306:3306"
    environment:
      MARIADB_ROOT_PASSWORD: root
    networks:
      - mlops_network
  adminer:
    image: adminer
    container_name: mariadb_adminer
    hostname: adminer
    restart: always
    ports:
      - "8089:8080"
    networks:
      - mlops_network
    networks:
      - mlops_network
      name: mlops_network
    external: true

```

**Bottom Panel:** A table showing port mappings. The '포트' (Port) column has two entries: 3306 and 8089. The '전달된 주소' (Published Address) column shows 'https://silver-waddle-7vx...'. The '실행 중인 프로세스' (Running Process) column shows '/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-po...'. The '표시 여부' (Display) column shows 'Private'. The '원본' (Source) column shows '자동 전달됨' (Automatically mapped). The row for port 8089 is highlighted with a red border.

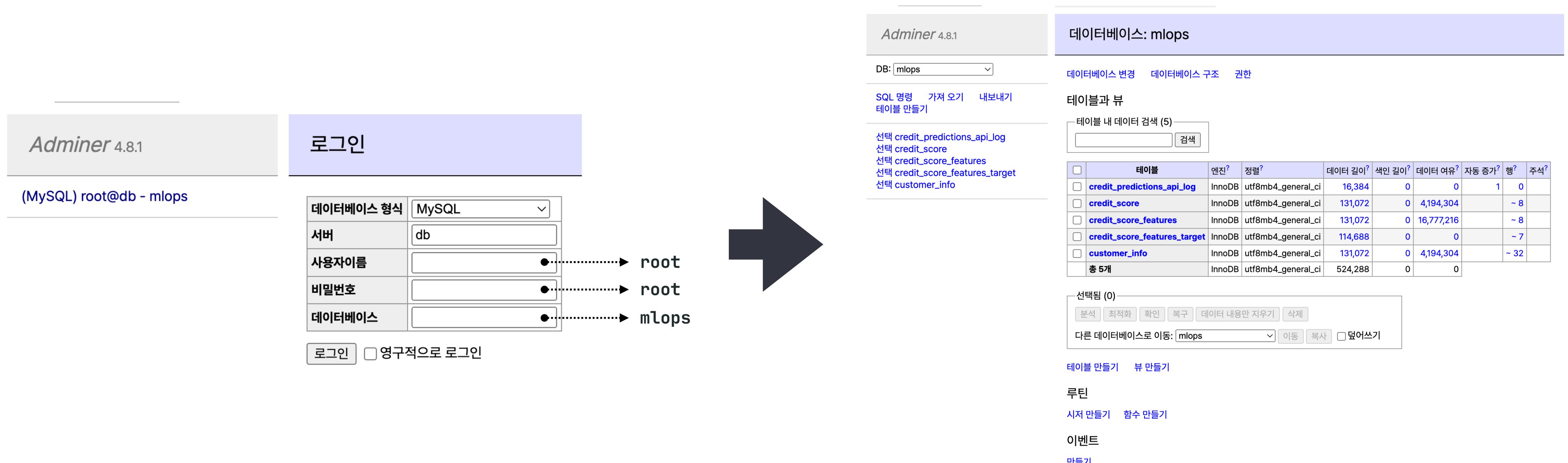
포트	전달된 주소	실행 중인 프로세스	표시 여부	원본
3306	https://silver-waddle-7vx...	/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-po...	Private	자동 전달됨
8089	https://silver-waddle-7vx...	/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-po...	Private	자동 전달됨

**Bottom Status Bar:**

- Codespaces: silver waddle
- main
- 0 ▲ 0
- Compose
- Jaeyoon Han (13시간 전) 줄 13, 열 19
- 공백: 2 LF {}
- 배열: U.S. ⌂

## 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

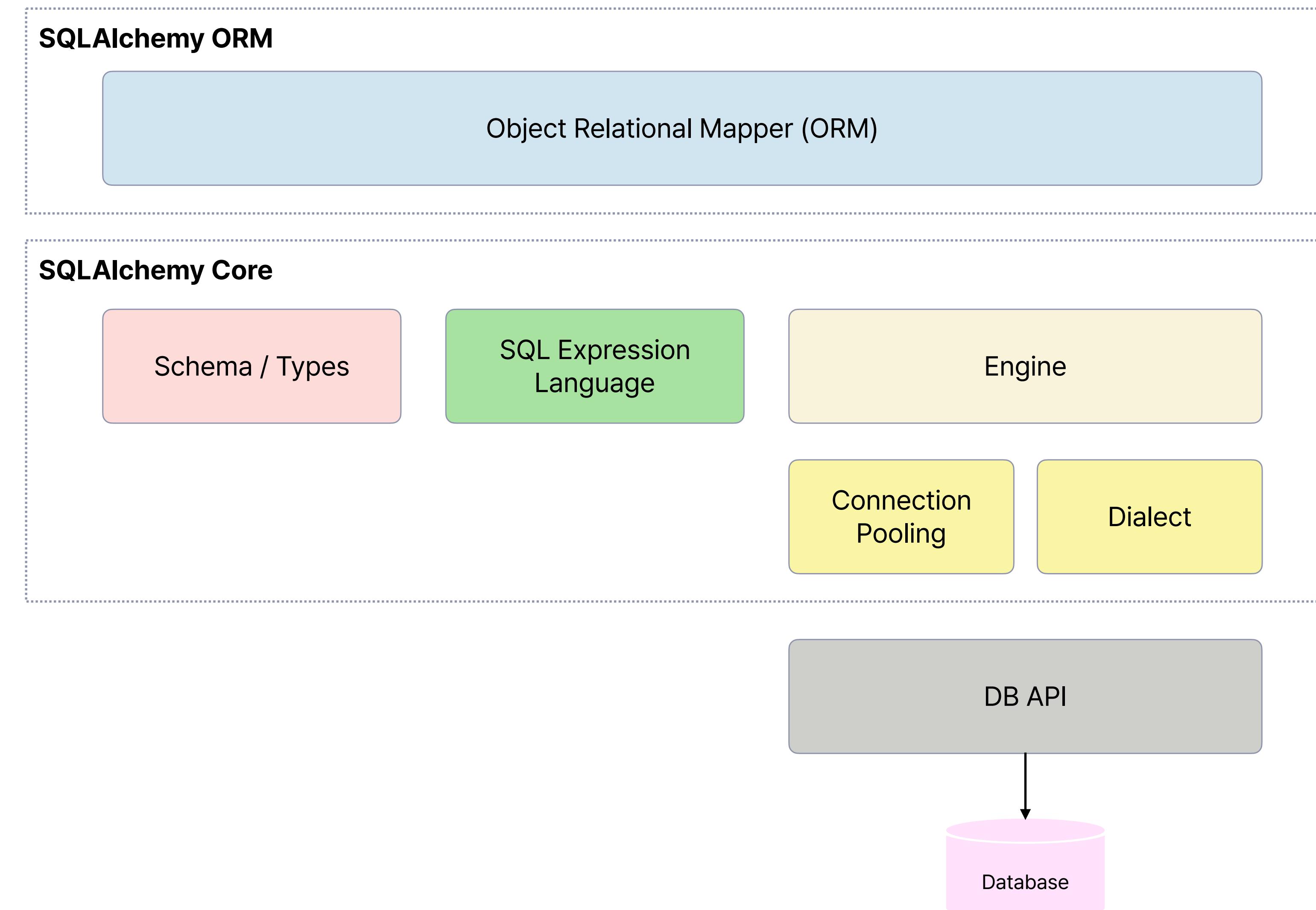
Python에서 가장 널리 사용되는 **ORM**

### 📌 SQLAlchemy 사용 시 주의점

- **Pandas와의 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

## 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(text(sql), con=conn)
```

## 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

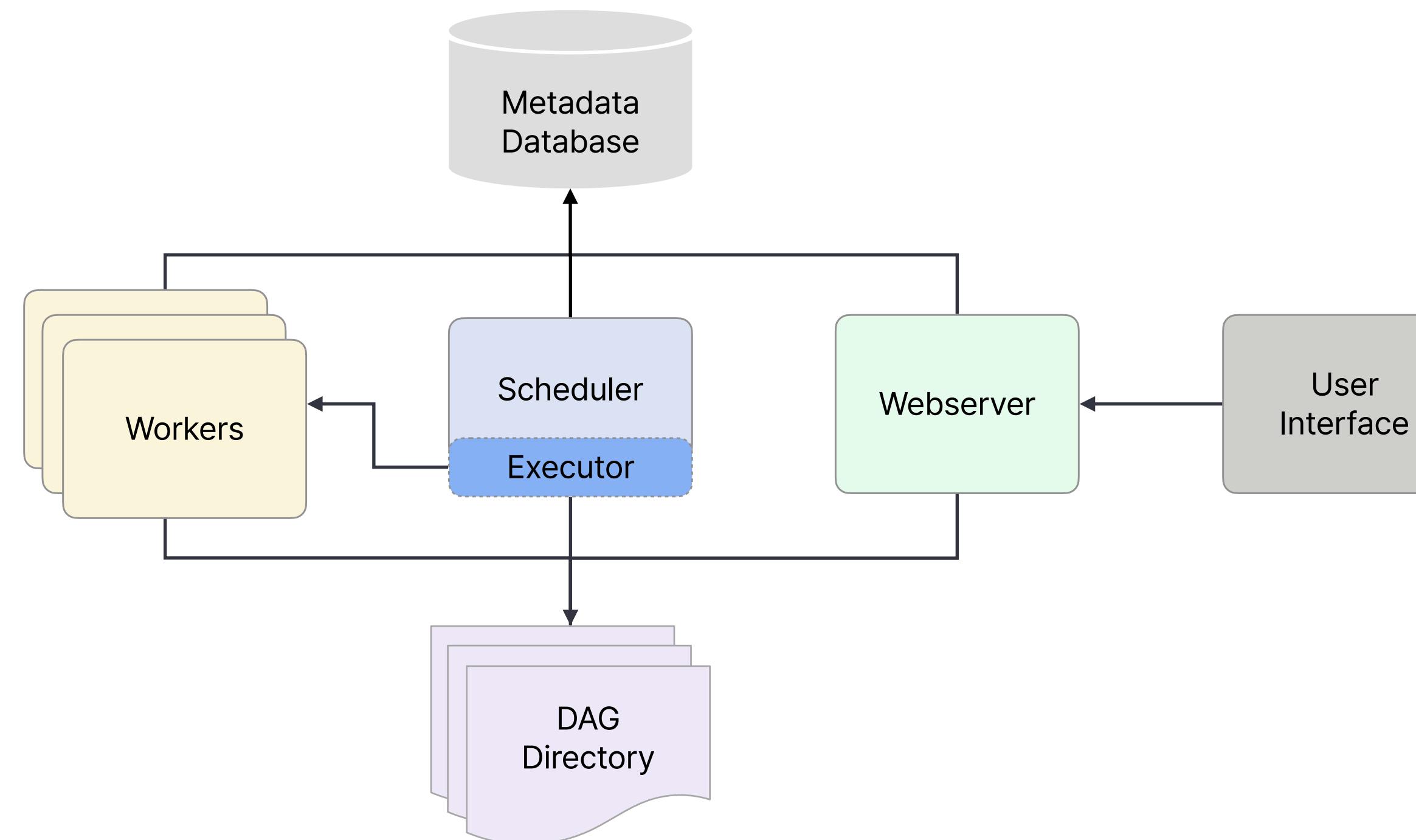
2

### Task

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

## 2.3 Airflow

# Airflow 아키텍처



**DAG Directory** • DAG를 보관하는 디렉토리

**Metadata Database** • Scheduler, Executor, Webserver에서 사용하는 메타데이터 저장 공간

**Worker** • 실제 Task를 처리하는 컴포넌트  
• Executor 종류에 따라 다른 방식으로 작동함

**Scheduler** • DAG의 실행을 계획하고, Task를 관리함  
• DAG가 실행될 때마다 실행할 Task를 정의하며, Task는 Executor에 따라 다른 방식으로 실행됨

**Executor** • Task의 실행 방법을 정의함  
• 다양한 Executor가 있어서 상황에 맞게 설정하여 사용

**Webserver** • DAG, 작업 및 작업 상태, 실행 로그 등을 표시하는 웹 인터페이스 제공

## 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"
$ airflow config get-value webserver enable_proxy_fix
# 마지막 실행 값이 False일텐데, 이 값을 True로 변경해줘야 함

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

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

# 가상환경 내에서 아래 명령어 실행 (source .venv/bin/activate 실행 후)
$ airflow db init
$ airflow standalone
```

### 2.3 Airflow

## Airflow 시작하기 – (1) 기본 설정

```
T), <TriggererHandlerWrapper (NOTSET)>
triggerer | [2025-02-04T06:21:44.549+0000] {triggerer_job_runner.py:338} INFO - Starting the triggerer
webserver | /workspaces/lgcns-advanced-mlops/.venv/lib/python3.11/site-packages/flask_limiter/extension.py:333 UserWarning: Using the in-memory storage for tracking
rate limits as no storage was explicitly specified. This is not recommended for production use. See: https://flask-limiter.readthedocs.io#configuring-a-storage-back
end for documentation about configuring the storage backend.
webserver | /workspaces/lgcns-advanced-mlops/.venv/lib/python3.11/site-packages/airflow/api_connexion/schemas/task_schema.py:52 ChangedInMarshmallow4Warning: `Numbe
r` field should not be instantiated. Use `Integer`, `Float`, or `Decimal` instead.
webserver | /workspaces/lgcns-advanced-mlops/.venv/lib/python3.11/site-packages/airflow/api_connexion/schemas/task_schema.py:55 ChangedInMarshmallow4Warning: `Numbe
r` field should not be instantiated. Use `Integer`, `Float`, or `Decimal` instead.
webserver | /workspaces/lgcns-advanced-mlops/.venv/lib/python3.11/site-packages/airflow/api_connexion/schemas/task_schema.py:59 ChangedInMarshmallow4Warning: `Numbe
r` field should not be instantiated. Use `Integer`, `Float`, or `Decimal` instead.
webserver | [2025-02-04 06:21:46 +0000] [19131] [INFO] Starting gunicorn 23.0.0
webserver | [2025-02-04 06:21:46 +0000] [19131] [INFO] Listening at: http://0.0.0.0:8080 (19131)
webserver | [2025-02-04 06:21:46 +0000] [19131] [INFO] Using worker: sync
webserver | [2025-02-04 06:21:46 +0000] [19171] [INFO] Booting worker with pid: 19171
webserver | [2025-02-04 06:21:46 +0000] [19172] [INFO] Booting worker with pid: 19172
webserver | [2025-02-04 06:21:46 +0000] [19173] [INFO] Booting worker with pid: 19173
webserver | [2025-02-04 06:21:46 +0000] [19177] [INFO] Booting worker with pid: 19177
standalone | Airflow is ready
standalone | Login with username: admin password: CxQxmkZU3Mhu3rkS
standalone | Airflow Standalone is for development purposes only. Do not use this in production!
[]
```

8080번 포트로 접속하여 Airflow에 로그인 진행

## 2.3 Airflow

# Airflow 시작하기 – (2) 계정 생성

The screenshot shows the Airflow web interface with the 'Security' menu item highlighted. A dropdown menu is open under 'Security' with the following options: List Users (highlighted with a red box), List Roles, User's Statistics, Actions, Resources, and Permissions. A large black arrow points downwards from the 'List Users' option towards the second screenshot.

**DAGs**

**List Users**

First Name	Last Name	User Name	Email	Is Active?	Role
Admin	User	admin	admin@example.com	True	[Admin]

## 2.3 Airflow

# Airflow 시작하기 – (2) 계정 생성

The screenshot shows the 'Add User' form in the Airflow web interface. The form fields and their corresponding mappings are:

- First Name \***: user → **user**
- Last Name \***: lgcns → **lgcns**
- User Name \***: user → **user**
- Is Active?**: checked → **user**
- Email \***: your.mail@gmail.com → **E-mail Address**
- Role \***: Admin → **Admin**
- Password \***: ..... → **user**
- Confirm Password \***: ..... → **user**

At the bottom left, there is a red-bordered **Save** button.

## 2.3 Airflow

# Airflow 시작하기 – (2) 계정 생성

Added Row

List Users

Search ▾

+ ← Record Count: 2

First Name	Last Name	User Name	Email	Is Active?	Role
Admin	User	admin	admin@example.com	True	[Admin]
user	lgcns	user	your.mail@gmail.com	True	[Admin]

해당 계정으로 다시 로그인하면 됨

### 2.3 Airflow

## Airflow 시작하기 – (3) 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 – (4) Airflow 변수 관리

The screenshot shows the Airflow web interface. At the top, there is a navigation bar with links: Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin (which is highlighted with a red box), and Docs. A dropdown menu for 'Admin' is open, showing options: Variables (also highlighted with a red box), Configurations, Connections, Plugins, Providers, Pools, and XComs. Below the navigation bar, there are two yellow warning boxes: one about not using SQLite as a metadata DB and another about not using the SequentialExecutor in production. A large black downward arrow is positioned in the center of the screen, pointing from the Admin menu towards the variable list below. The main content area is titled 'DAGs' and shows a 'List Variable' section. It includes a '파일 선택' button, a file selection input field showing '선택된 파일 없음', and three radio buttons for file handling: 'Overwrite if exists' (selected), 'Fail if exists', and 'Skip if exists'. There is also an 'Import Variables' button with a cloud icon. The 'List Variable' section has a search bar and a toolbar with a '+' button (highlighted with a red box), an 'Actions' dropdown, and a back arrow. It displays a message 'No records found' and a 'Record Count: 0'.

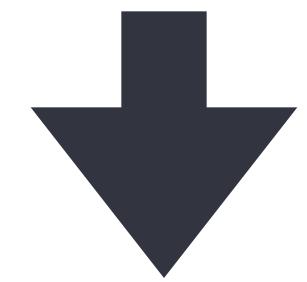
## 2.3 Airflow

# Airflow 시작하기 – (4) Airflow 변수 관리

Add Variable

Key *	AIRFLOW_DAGS_PATH
Val	/workspaces/advanced-mlops
Description	

**Save** [←](#)



Record Count: 1

	Key ↑	Val ↑	Description ↑	Is Encrypted ↑
<a href="#">Actions</a> <a href="#">←</a>	AIRFLOW_DAGS_PATH	/workspaces/advanced-mlops		False

```
airflow variables get AIRFLOW_DAGS_PATH
```

## 2.3 Airflow

# 간단한 DAG 만들어보기

pipelines/tutorial/first\_dag.py

```
with DAG(  
    [REDACTED]  
) as dag:  
    task1 = BashOperator(  
        [REDACTED]  
    )  
    task2 = BashOperator(  
        [REDACTED]  
    )  
    Task3 = BashOperator(  
        [REDACTED]  
    )  
  
    task1 >> [task2, task3]
```

DAG에 대한 여러 파라미터를 작성

Task를 정의하는 Operator에 맞는 파라미터를 추가

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=["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(
    """
    {% for i in range(5) %}
        echo "ds = {{ ds }}"
        echo "macros.ds_add(ds, {{ i }}) = {{ macros.ds_add(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

Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. [Click here](#) for more information.

Do not use the **SequentialExecutor** in production. [Click here](#) for more information.

## DAGs

Do not use **SQLite** as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. [Click here](#) for more information.

Do not use the **SequentialExecutor** in production. [Click here](#) for more information.

## DAGs

All 1	Active 0	Paused 1	Running 0	Failed 0	x lgcns	Search DAGs	Auto-refresh	C
<i>i</i> DAG <i>▼</i> <input checked="" type="checkbox"/> simple_dag lgcns mlops	Owner <i>▼</i> user	Runs <i>i</i> 	Schedule 0 15 * * *	Last Run <i>▼</i> <i>i</i> 2025-03-01, 15:00:00	Next Run <i>▼</i> <i>i</i> 	Recent Tasks <i>i</i> 	Actions   	Links

« ‹ 1 › ›» Showing 1-1 of 1 DAG

## 2.3 Airflow

# 간단한 DAG 만들어보기

## DAGs

All (1) Active (1) Paused (0)

Running (0) Failed (0)

x lgcns

Search DAGs

Auto-refresh  Refresh

DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
<b>simple_dag</b> lgcns mlops	user	1	0 15 * * *	2025-02-04, 17:51:27	2025-03-01, 15:00:00	1	<input type="button" value="▶"/> <input type="button" value="↻"/> <input type="button" value="trash"/>	...

Showing 1-1 of 1 DAGs

Search ▾

Actions   Record Count: 1

	State	Dag Id	Logical Date	Run Id	Run Type	Queued At	Start Date	End Date	Note	External Trigger	Conf	Duration
<input type="checkbox"/>	<b>success</b>	simple_dag	2025-02-04, 17:51:27	manual_2025-02-04T08:51:27.107500+00:00	manual	2025-02-04, 17:51:27	2025-02-04, 17:51:28	2025-02-04, 17:51:28	True	{}	<1s	

## 2.3 Airflow

# 간단한 DAG 만들어보기

**DAG: simple\_dag** Simple airflow dag

Schedule: 0 15 \* \* \* | Next Run ID: 2025-03-01, 15:00:00 KST | Auto-refresh | 25 | Press shift + / for Shortcuts

Duration: 00:00:00

print\_date  
sleep  
print\_with\_loop

**DAG simple\_dag**

**DAG Runs Summary**

- Total Runs Displayed: 1
- Total success: 1
- First Run Start: 2025-02-04, 17:51:28 KST
- Last Run Start: 2025-02-04, 17:51:28 KST
- Max Run Duration: 00:00:00
- Mean Run Duration: 00:00:00
- Min Run Duration: 00:00:00

**DAG Summary**

- Total Tasks: 3
- BashOperators: 3

**DAG Details**

- Dag display name: simple\_dag
- Dag id: simple\_dag
- Description: Simple airflow dag
- Fileloc: /Users/jayhan/airflow/dags/pipelines/tutorial/first\_dag.py



# 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 illustrates the process of navigating to the 'Connections' page in the Airflow Admin interface.

**Top Navigation:** The top navigation bar includes links for Airflow, DAGs, Cluster Activity, Datasets, Security, Browse, Admin (which is highlighted with a red box), and Docs.

**Admin Submenu:** A dropdown menu is open under the Admin link, listing Variables, Configurations, Connections (which is also highlighted with a red box), Plugins, Providers, Pools, and XComs.

**Large Arrow:** A large black arrow points downwards from the Admin submenu towards the 'List Connection' page below.

**List Connection Page:** The bottom section shows the 'List Connection' page. It features a search bar, action buttons (+, Actions, and back/forward arrows), and a table with 61 records. The table columns are Conn Id, Conn Type, Description, Host, Port, Is Encrypted, and Is Extra Encrypted. The data rows are:

	Conn Id	Conn Type	Description	Host	Port	Is Encrypted	Is Extra Encrypted
<input type="checkbox"/>	airflow_db	mysql		mysql		False	False
<input type="checkbox"/>	athena_default	athena				False	False
<input type="checkbox"/>	aws_default	aws				False	False

### 3.1 데이터 추출

## Airflow 추가 설정

Add Connection

Connection Id \*  → **feature\_store**

Connection Type \*  → **MySQL**  
Connection Type missing? Make sure you've installed the corresponding Airflow Provider Package.

Description

Host  → **0.0.0.0**

Schema  → **mlops**

Login  → **root**

Password  → **root**

Port  → **3306**

Extra

**Save** **Test** ←

### 3.1 데이터 추출

## 데이터 병합

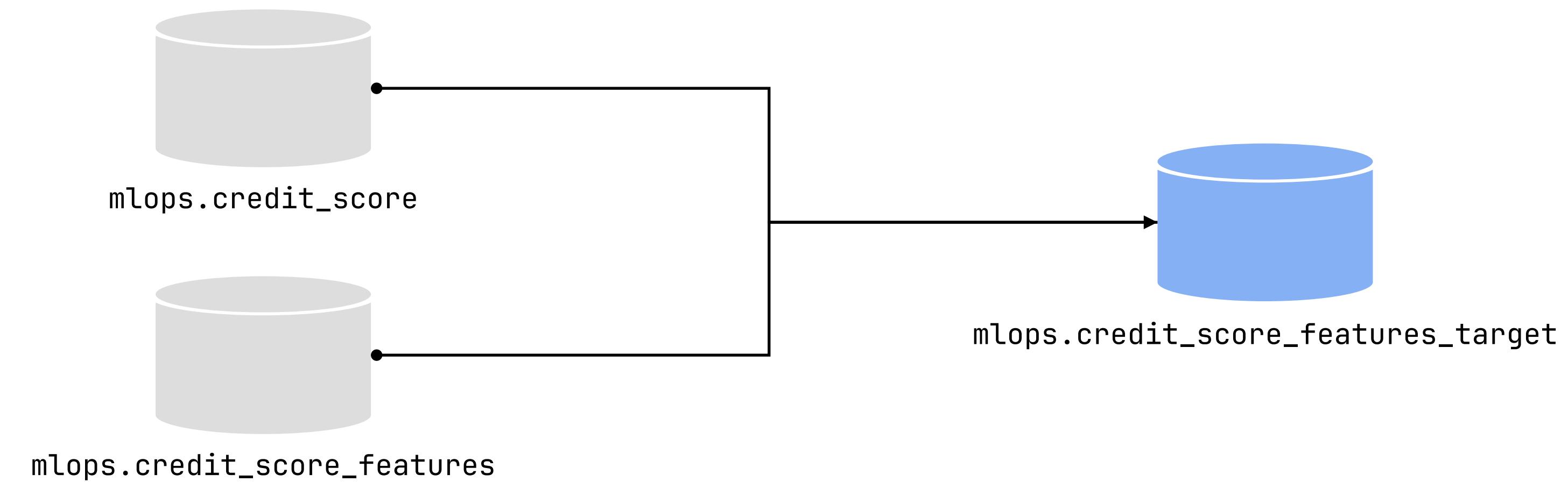
데이터베이스: mlops

데이터베이스 변경 데이터베이스 구조 권한

테이블과 뷰

테이블 내 데이터 검색 (4)

	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('{{ ds }}', '%Y-%m-%d'), ● → Ninja Template
        INTERVAL -7 DAY
    ),
    '%Y-%m-%d'
) OR
base_dt = STR_TO_DATE('{{ ds }}', '%Y-%m-%d'); ● → Ninja Template
```

### 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('{{ ds }}', '%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('{{ ds }}', '%Y-%m-%d'), •————→ Jinja Template
        INTERVAL -1 MONTH
    )
    AND STR_TO_DATE('{{ ds }}', '%Y-%m-%d') •————→ Jinja Template
) b ON a.id = b.id
    AND a.customer_id = b.customer_id;
```

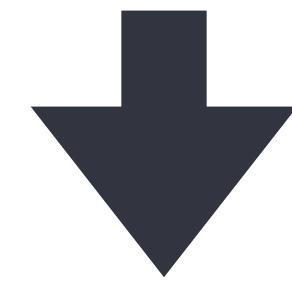
### 3.1 데이터 추출

## Task 개발

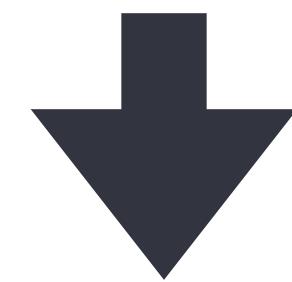
---

**EmptyOperator**

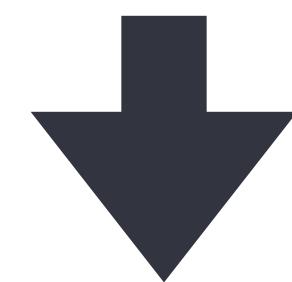
아무 작업도 하지 않는 Task로 DAG의 큰 틀을 구성



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=["lgcns", "mllops"],
) 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  
)
```

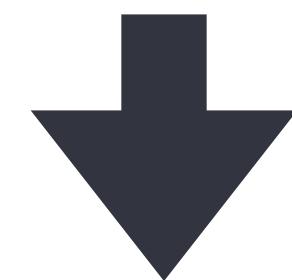
### 3.1 데이터 추출

## DAG 실행 결과 확인

### DAGs

The screenshot shows the Airflow interface for managing Data Pipelines (DAGs). At the top, there are filters for 'All' (2), 'Active' (2), and 'Paused' (0) DAGs, along with counts for 'Running' (0) and 'Failed' (0) runs. A search bar is present, and there are buttons for 'Auto-refresh' and a refresh icon. The main table lists two DAGs:

- credit\_score\_classification\_ct**: Paused, Owner: user, Schedule: None. Recent Tasks: 0.
- simple\_dag**: Active, Owner: user, Schedule: 0 15 \* \* \*. Last Run: 2025-02-04, 17:51:27. Next Run: 2025-03-01, 15:00:00. Recent Tasks: 1.

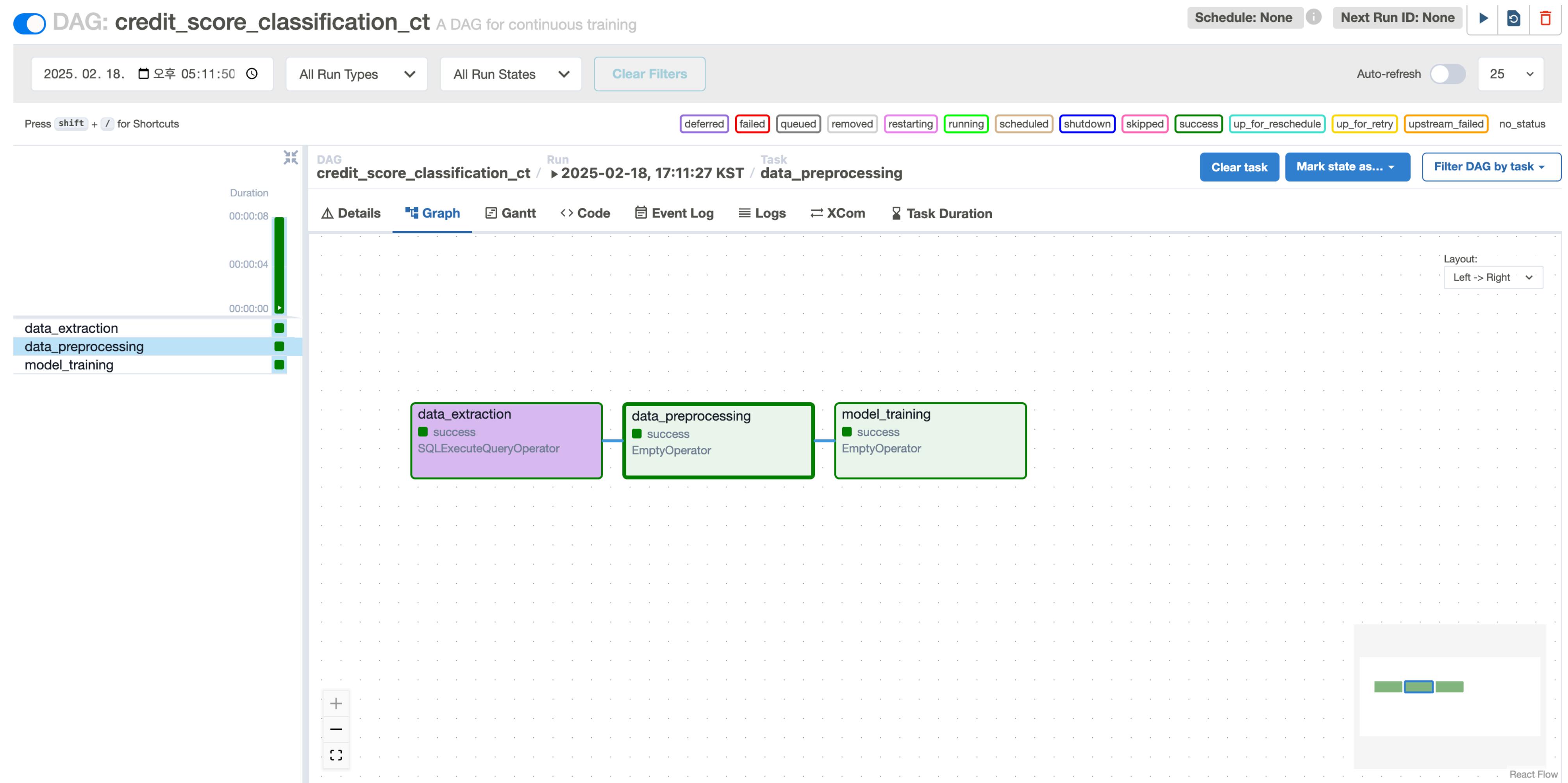


### DAGs

This screenshot shows the same Airflow interface after some time has passed. The 'credit\_score\_classification\_ct' DAG is now active, indicated by a blue switch icon, and has one recent task run on 2025-02-05 at 20:58:42. The 'simple\_dag' DAG remains active with one recent task run on 2025-02-04 at 17:51:27. The rest of the interface and table structure remain consistent with the first screenshot.

### 3.1 데이터 추출

## DAG 실행 결과 확인



## **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**

```
.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)
- 각 CSP에서 제공하는 기능 활용

### 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 수정
    - 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
drwxr-xr-x@ - jayhan 5 Feb 11:12 data_extract
.rw-r--r--@ 0 jayhan 5 Feb 10:43 __init__.py
.rw-r--r--@ 2.1k jayhan 5 Feb 20:58 features.sql
drwxr-xr-x@ - jayhan 6 Feb 10:40 data_preprocessing
.rw-r--r--@ 0 jayhan 6 Feb 10:40 __init__.py
.rw-r--r--@ 9.9k jayhan 7 Feb 10:43 preprocessor.py
drwxr-xr-x@ - jayhan 7 Feb 10:41 docker
.rw-r--r--@ 738 jayhan 7 Feb 14:43 Dockerfile → Dockerfile
.rw-r--r--@ 687 jayhan 7 Feb 16:37 docker-compose.yml → 위 Dockerfile을 이미지로 하는 컨테이너를 띄우기 위한 설정 파일
.rw-r--r--@ 117 jayhan 7 Feb 14:49 requirements.txt → 컨테이너 내에서 pip로 설치할 라이브러리 목록
drwxr-xr-x@ - jayhan 5 Feb 10:41 training
.rw-r--r--@ 0 jayhan 5 Feb 16:58 __init__.py
.rw-r--r--@ 2.0k jayhan 7 Feb 16:53 continuous_training_dag.py
drwxr-xr-x@ - jayhan 4 Feb 17:17 tutorial
.rw-r--r--@ 0 jayhan 4 Feb 17:17 __init__.py
.rw-r--r--@ 1.4k jayhan 4 Feb 17:15 first_dag.py
.rw-r--r--@ 0 jayhan 4 Feb 17:17 __init__.py
```

### 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:3.11-slim
LABEL maintainer="otzslayer@gmail.com"

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

USER codespace

RUN mkdir -p ${USER_HOME}/artifacts \
    && pip install --no-cache-dir \
    -r ${USER_HOME}/requirements.txt

WORKDIR ${USER_HOME}
```

`pipelines/continuous_training/docker/Dockerfile`

File/Folder	Type	Owner	Size	Last Modified
.	Folder	jayhan	-	8 Feb 13:04
data_preprocessing	Folder	jayhan	-	8 Feb 13:04
preprocessor.py	File	jayhan	9.9k	8 Feb 13:04
utils	Folder	jayhan	-	8 Feb 13:04
__init__.py	File	jayhan	0	8 Feb 13:04
callbacks.py	File	jayhan	194	8 Feb 13:04
common.py	File	jayhan	156	8 Feb 13:04
dates.py	File	jayhan	872	8 Feb 13:04
requirements.txt	File	jayhan	117	8 Feb 13:04

### 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.operators.bash import BashOperator

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": "{{ ds }}",
    },
    append_env=True, ● -----> 위 값을 기준 환경변수를 대체하지 않고 추가
    retries=1,
)
```

### 3.2 데이터 전처리

## Task 개발

---

### 폴더 권한에 대한 고민

- 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

**DAG: credit\_score\_classification\_ct** A DAG for continuous training

2025. 02. 18. 오후 05:13:40 All Run Types All Run States Clear Filters Schedule: None Next Run ID: None Auto-refresh 25

Press shift + / for Shortcuts deferred failed queued removed restarting running scheduled shutdown skipped success up\_for\_reschedule up\_for\_retry upstream\_failed no\_status

DAG credit\_score\_classification\_ct / Run Task credit\_score\_classification\_ct / data\_preprocessing

Clear task Mark state as... Filter DAG by task

Duration

00:01:25  
00:00:42  
00:00:00

data\_extraction  
data\_preprocessing  
model\_training

Details Graph Gantt Code Event Log Logs XCom Task Duration

All Levels All File Sources Wrap Download See More

```
[2025-02-18, 17:14:54 KST] {subprocess.py:106} INFO - #15 exporting layers
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - #15 exporting layers 5.3s done
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - #15 writing image sha256:0f81ec2c7201d3281fd2a7247a93d7e8dc5595d937304225306ea7bfa8039036 done
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - #15 naming to docker.io/library/credit_score_classification:ct-pipeline-latest 0.0s done
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - #15 DONE 5.3s
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - #16 [continuous_training_pipeline] resolving provenance for metadata file
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - #16 DONE 0.0s
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - continuous_training_pipeline Built
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - Container credit_score_classification_ct_pipeline Creating
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - Container credit_score_classification_ct_pipeline Created
[2025-02-18, 17:14:59 KST] {subprocess.py:106} INFO - Attaching to credit_score_classification_ct_pipeline
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to age.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to annual_income.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to monthly_inhand_salary.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to num_bank_accounts.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to num_credit_card.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to interest_rate.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to num_of_loan.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to delay_from_due_date.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to num_of_delayed_payment.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to changed_credit_limit.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to num_credit_inquiries.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to outstanding_debt.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to credit_utilization_ratio.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to credit_history_age.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to total_emis_per_month.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to amount_invested_monthly.
[2025-02-18, 17:15:02 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline | RobustScaler has been applied to monthly_balance.
[2025-02-18, 17:15:03 KST] {subprocess.py:106} INFO - credit_score_classification_ct_pipeline exited with code 0
[2025-02-18, 17:15:03 KST] {subprocess.py:106} INFO - Container credit_score_classification_ct_pipeline Stopping
[2025-02-18, 17:15:03 KST] {subprocess.py:106} INFO - Container credit_score_classification_ct_pipeline Stopped
[2025-02-18, 17:15:03 KST] {subprocess.py:106} INFO - Container credit_score_classification_ct_pipeline Removing
[2025-02-18, 17:15:03 KST] {subprocess.py:106} INFO - Container credit_score_classification_ct_pipeline Removed
[2025-02-18, 17:15:03 KST] {subprocess.py:106} INFO - Command exited with return code 0
[2025-02-18, 17:15:03 KST] {taskinstance.py:340} ▶ Post task execution logs
```

### **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

## 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=mlops:mlops ${UTIL_PATH}/ ${USER_HOME}/utils  
COPY --chown=mlops:mlops ${PREPROCESSING_PATH}/preprocessor.py \  
    ${USER_HOME}/data_preprocessing/  
COPY --chown=mlops:mlops ${TRAINING_PATH}/trainer.py \  
    ${USER_HOME}/training/  
COPY --chown=mlops:mlops ${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.operators.bash import BashOperator

data_preprocessing = BashOperator(
    task_id="model_training",
    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/training/trainer.py",
        "MODEL_NAME": "credit_score_classification",
        "BASE_DT": "{{ ds }}",
    },
    append_env=True, ● → 위 값을 기준 환경변수를 대체하지 않고 추가
    retries=1,
)
```



# 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

### 데이터 모델 개발

1

- 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

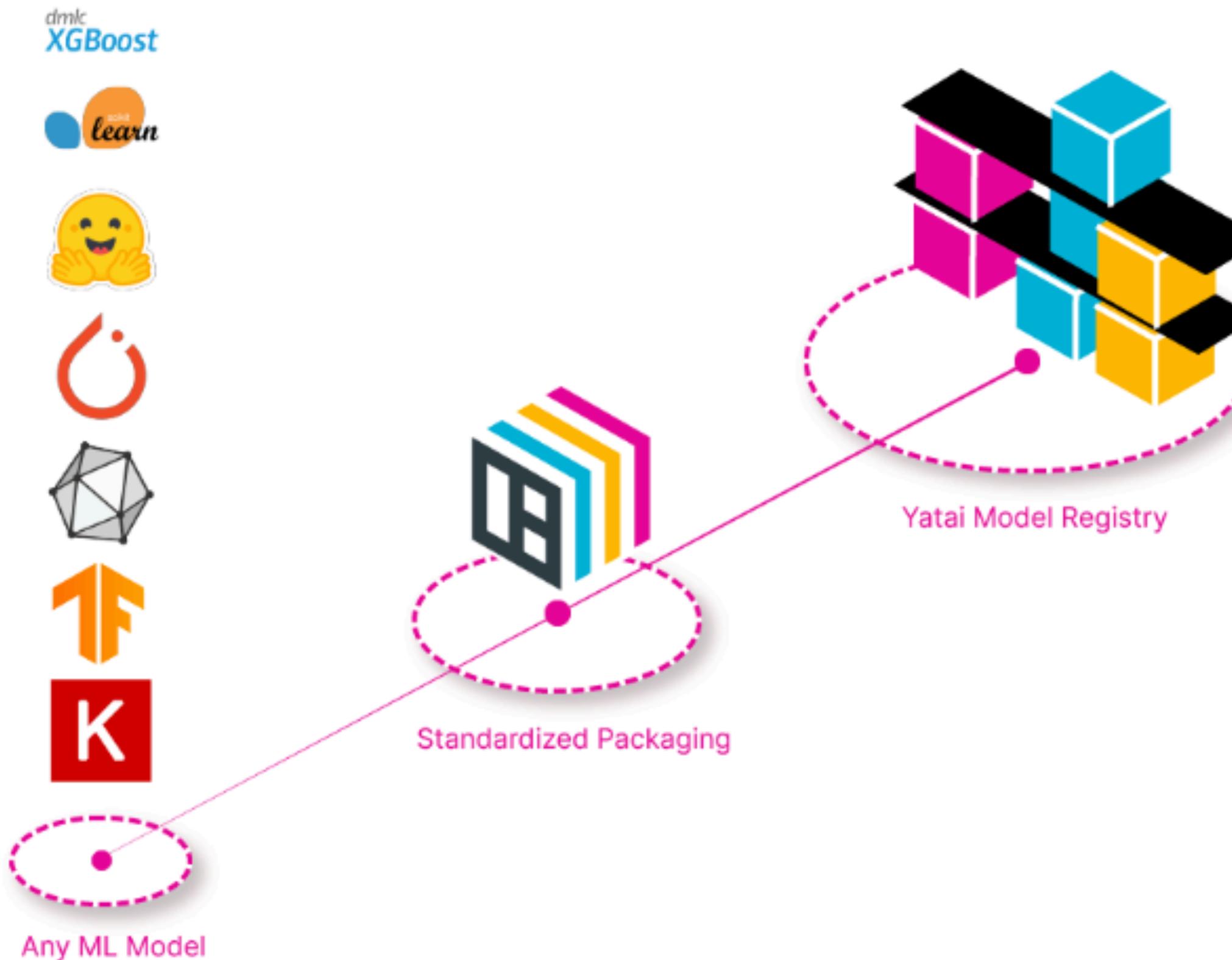
5

### 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 ! bentofile.yaml
                           → 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")

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

def get_db():
    """데이터베이스 세션을 제공하는 함수 (의존성 주입용)"""
    with SessionLocal() as session:
        yield session
```

api/src/db.py

## sessionmaker

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

## sessionmaker의 필요성

### 1. DB 연결을 관리

- SQLAlchemy에서는 데이터베이스와 직접 연결하지 않고,  
**세션(session)**을 통해 데이터를 읽고 쓰는 작업을 하기 때문

### 2. 일관된 세션 생성

- sessionmaker를 사용하면 동일한 설정을 가진 세션을 반복해서 쉽게 만들 수 있음

### 3. 트랜잭션 관리

- session.commit() 또는 session.rollback()을 사용하여  
데이터 변경 사항을 반영하거나 되돌릴 수 있음

## 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, bind=engine)
Base = declarative_base()
```

```
def get_db():
    """데이터베이스 세션을 제공하는 함수 (의존성 주입용)"""
    with SessionLocal() as session:
        yield session
```

## 의존성 주입

- API가 직접 세션을 생성하지 않고  
외부에서 세션을 받아 유연성을 높이고 결합도를 낮춤

## Context Manager를 이용한 세션 관리

- with절과 같은 context manager를 사용하면  
세션을 사용한 다음 자동으로 세션을 닫을 수 있음

#### 4.1 모델 배포 (API 개발)

## 테이블 모델

---

api/src/models.py

```
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, db: Session = next(get_db())):
        ●→ 세션을 의존성 주입하여 사용
        self.db = db
        self.bento_model = bentoml.models.get("credit_score_classifier:latest") ●→ 저장한 모델 불러오기
        self.robust_scalers = joblib.load(●→ 전처리 시 저장한 RobustScaler 객체 불러오기
            os.path.join(encoder_path, "robust_scaler.joblib"))
    )
    self.model = bentoml.catboost.load_model(self.bento_model)

```

## 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")

    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.item(),
        features=data.model_dump(),
        prediction=label,
        confidence=prob,
        elapsed_ms=elapsed_ms,
    )
    self.db.add(record) •→ 테이블에 데이터 추가 후 아래에서 커밋 진행
    self.db.commit()
    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 services:CreditScoreClassifier
```

## 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>Download</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)

# Docker 설정 파일 개발

## 1 Dockerfile

```
api/docker/Dockerfile
FROM python:3.11-slim

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

RUN apt-get update && apt-get install -y git && rm -rf /var/lib/apt/lists/*

USER codespace

RUN pip install --no-cache-dir \
--trusted-host pypi.org --trusted-host files.pythonhosted.org \
-r ${USER_HOME}/requirements.txt

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

WORKDIR ${USER_HOME}/${API_PATH}
```

## 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

---

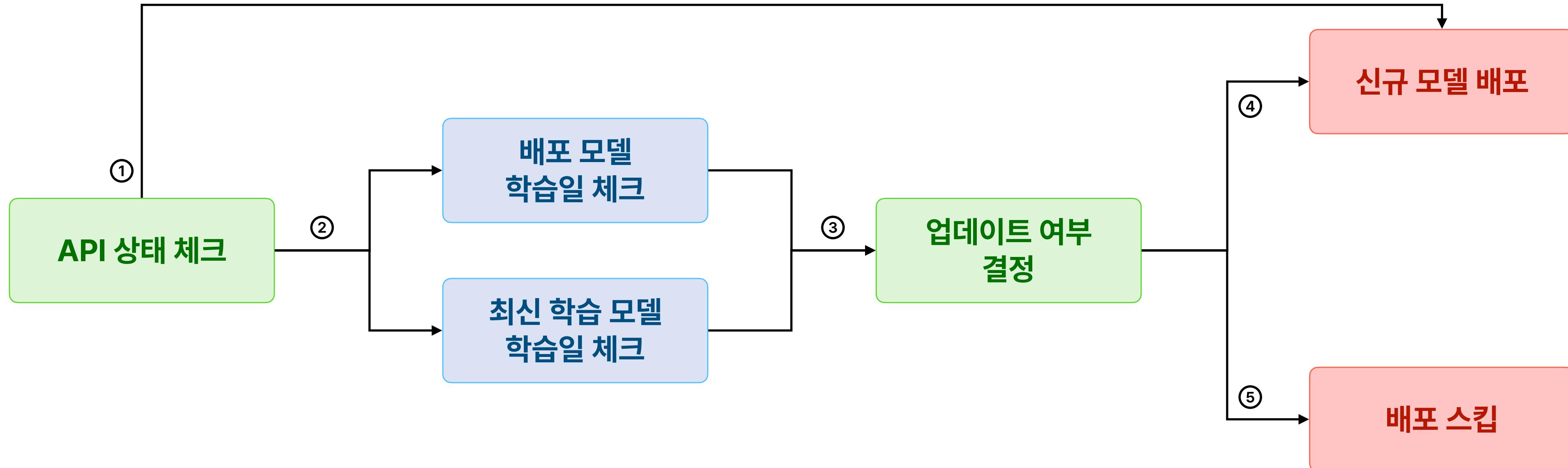
```
# api 폴더로 이동하여 Docker로 API 서비스 실행 (백그라운드 실행)
$ cd api
$ docker compose up -d
```

## **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 반환**해야 함
- 선택된 Task만 실행되며, 선택되지 않은 Task는 Skipped 상태가 됨
- 인자로 `provide_context` 추가하여 다음 실행할 Task 정보를 보냄

## 4.2 지속적 배포 구현

# PythonOperator vs BranchPythonOperator

**DAG: credit\_score\_classification\_cd** A DAG for continuous deployment

2025. 02. 14. 오후 02:37:51

All Run Types All Run States Clear Filters

Schedule: None Next Run ID: None Auto-refresh 25

Press shift + / for Shortcuts

Duration

00:00:23

00:00:11

00:00:00

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

Deferred Failed Queued Removed Restarting Running Scheduled Shutdown Skipped Success Up\_for\_reschedule Up\_for\_retry Upstream\_failed No\_status

**DAG credit\_score\_classification\_cd**

Details Graph Gantt Code Event Log Run Duration Task Duration Calendar

**DAG Runs Summary**

Total Runs Displayed 2

Total success 2

First Run Start 2025-02-13, 12:09:38 KST

Last Run Start 2025-02-13, 12:31:16 KST

Max Run Duration 00:00:23

Mean Run Duration 00:00:19

Min Run Duration 00:00:15

# **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 !**