

MLOps Engineering

Machine Learning Operations V2.0.0

Sessions 12 – 13 (WIP)

MsC in Business Analytics and Data Science

Madrid, Jun 2025

Agenda

- **1st Group exec summary (Q&A)**
- **CI/ CD with GitHub Actions**
- **FastAPI for model serving**
- **Docker for deployment**

Preprocessing Performance vs. Data Validation

In your team's modular MLOps pipeline, during a code review, you notice the preprocessing step now takes twice as long due to extensive data validation checks, such as checking for missing values, data types, and outliers. Your junior insists these validations are critical for ensuring data quality. What decision do you make?

Set up the options. Choose one or multiple correct answers.

☐ A Keep all validations; accuracy outweighs performance

☒ B Move validation steps to a parallel process asynchronously. Leveraging cloud parallel processing

☒ C Reduce validations only to those that protect data integrity, enhancing performance

Correct answer

☐ D Remove most checks, and rely on the Pytest suite (that's what is there for)

Code Comments

In your team's modular MLOps pipeline, your junior has added extensive comments to the code, explaining what each function does, such as "this function loads the data." However, they haven't included comments explaining why certain strategies or parameters were chosen, like why a specific batch size was selected. How do you guide them to improve their documentation?

Set up the options. Choose one or multiple correct answers.

☐ A Comments explaining "what" are sufficient for maintainability

☒ B Replace code-level comments with a comprehensive README.md

☒ C Emphasize adding brief but clear explanations of "why" key decisions were made

Correct answer

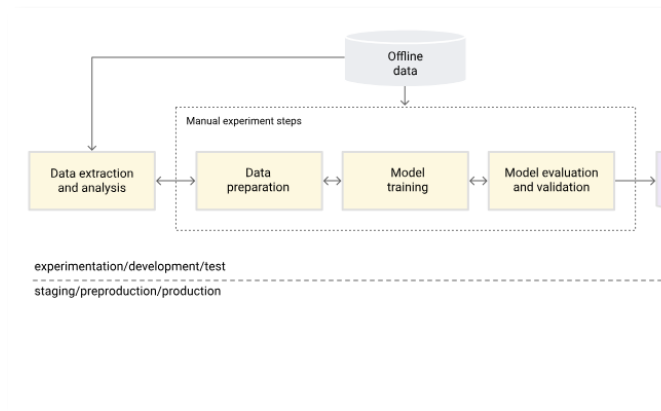
☐ D Suggest adding links to external documentation for any unclear choices

Details for grading tests for 1st Group assignment

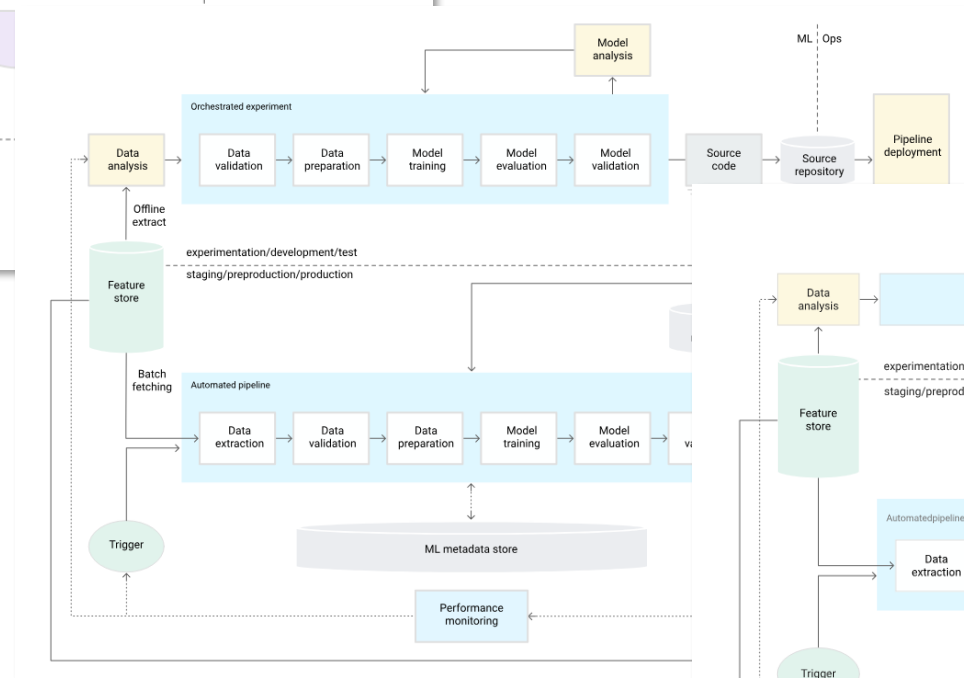
Test / Command	Purpose	Rubric dimension	Grading logic	Recommendation
black --check .	PEP 8 formatting	Code Quality & Efficiency	Score = (num_compliant_files / total_files) × 10	Run black . to auto-format the code
isort --check-only .	Import order style	Code Quality & Efficiency	Score = (num_compliant_files / total_files) × 10	Run isort . to auto-format all imports
flake8 .	Linting for style and simple bugs	Code Quality & Efficiency	Relative count of non-compliant across groups	Action on console feedback Use auto-formatters (black, isort)
flake8 --select=LOG <path>	Static check for correct logging usage (no f-strings, placeholders, etc.) ([github.com][1])	Logging & Monitoring	Relative count of non-compliant across groups	Action on console feedback
pylint src --score y	Advanced linting, docstrings score	Code Quality & Documentation	Direct pylint overall score	Action on console feedback
pytest --cov=src --cov-report=term-missing	Unit tests and coverage	Testing & Coverage	1. % number of tests that pass, vs. total 2. test coverage	Action on console feedback
radon cc -s -a src	Average cyclomatic complexity	Modularization & Code Quality	Map Radon average: A→10, B→8, C→6, D→4, E→2, F→0	Refactor high-complexity functions
detect-secrets scan --exclude-files '\.ipynb\$'	Detect hard-coded secrets	Security	10/10 if no secrets on src code	Review and remove any secrets; use environment variables or .env files
pydocstyle src	Docstring presence and style	Documentation & Clarity	Violation/ files normalised x-groups	Action on console feedback
File presence check (custom Python)	Confirm key project files/folders exist (README.md, config.yaml, environment.yml, tests/, .gitignore, logs/, models/)	Documentation, Config & Artifacts	(files_found/expected) × 10	Ensure full config mgmt. logging and artifacting completeness
Config usage scan (custom Python)	Count hard-coded absolute paths vs use of os.environ / yaml configs	Config Mgmt	max(0, 10 – hardcoded_paths)	Ensure no paths are hard-coded
Error-handling scan (custom Python)	Count try: blocks and custom Exception subclasses	Error Handling & Validation	min(10, try_blocks ÷ 10 × 10)	If low, increase use of try/except blocks and define custom exceptions

MLOps is a journey - involving people, processes, tools and data transformation

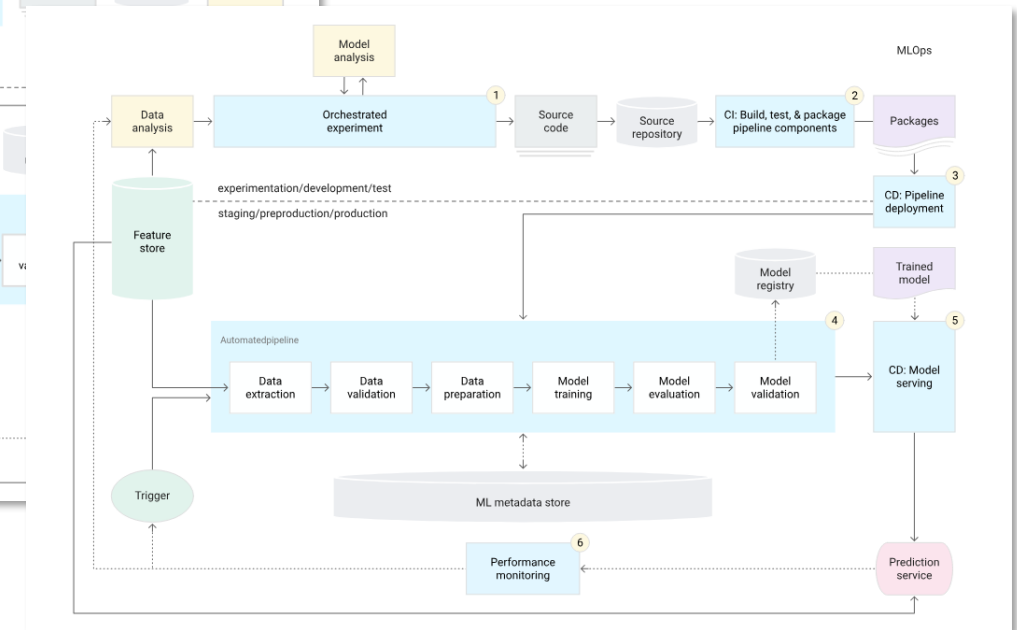
MLOps level 0: Manual process



MLOps level 1: ML pipeline automation

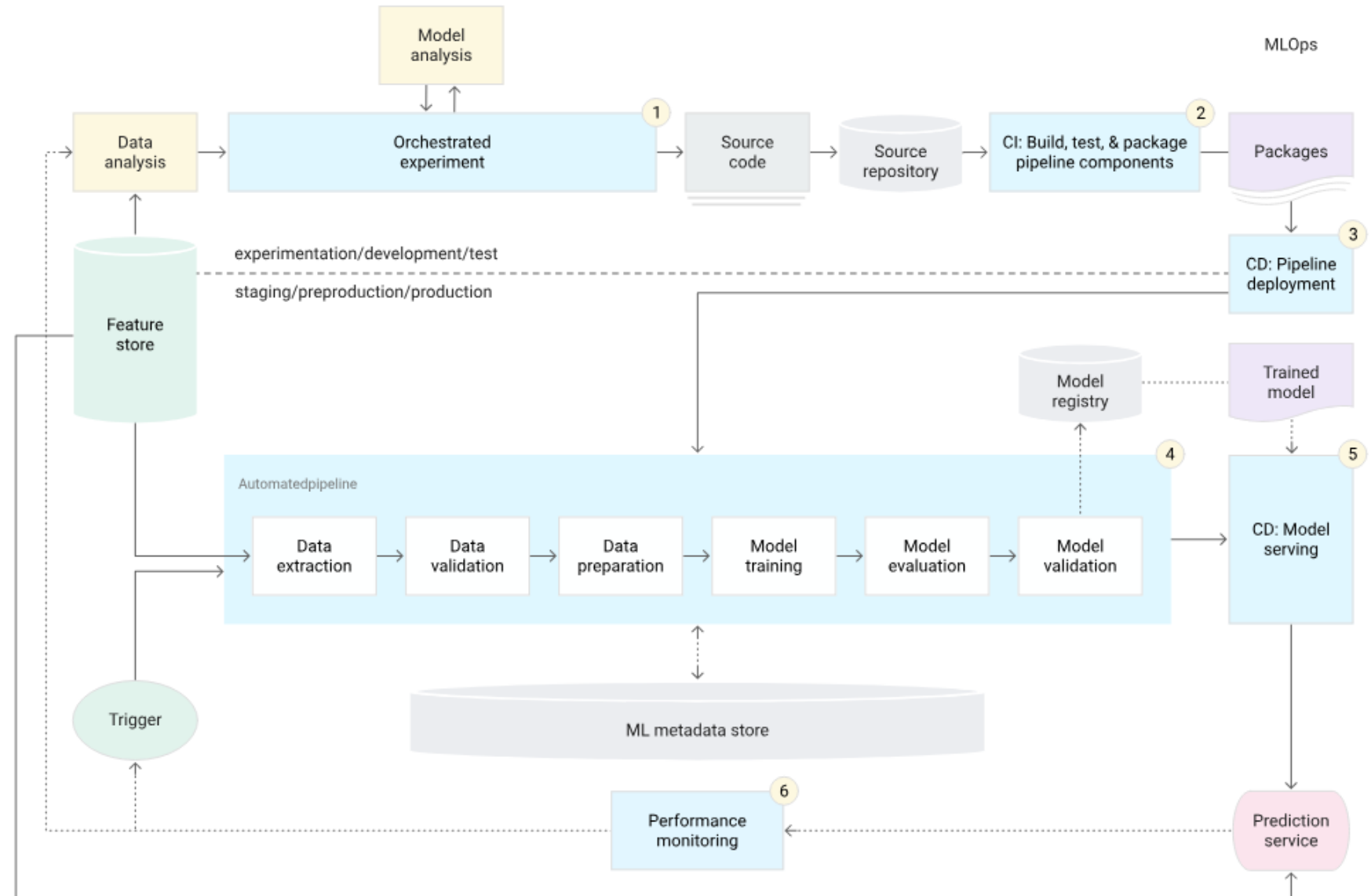


MLOps level 2: CI/CD pipeline automation

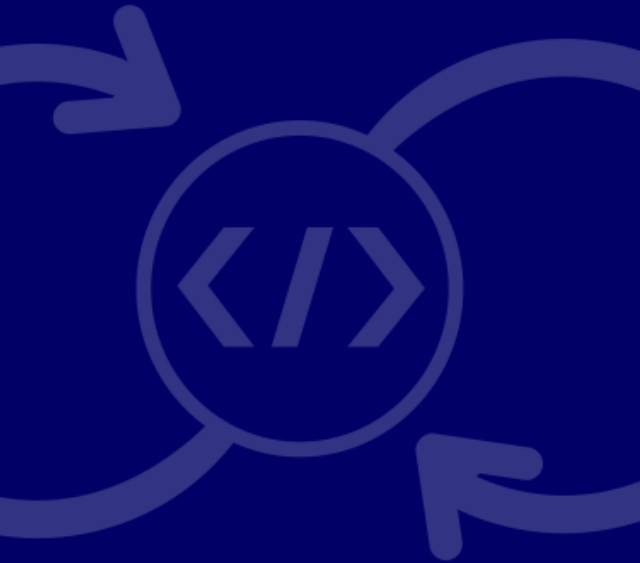


Level 2

Automate CI/CD pipelines to reliably retrain and deploy ML models as data and business needs evolve fast



Continuously integrate and verify code to catch errors early, then deliver **only tested, deployable builds** to production



Continuous Integration (CI)

Refers to continuous building of the code/ model. Automatically triggered when:

- New code is pushed (or committed)
- Runs the test suit (Unit and Integration)
- Compiles the code after pushing the changes
- Builds packages, containers, executables, etc.
- Delivers the code to the CD pipeline

Continuous Delivery (CD)

Ensures the code is “always” deployable:

- Code/ model is verified by the CI tasks
- Ensures compatibility of code/ models with target environment
- Checks model performance before deploying
- Code is verified by a Human
- Code/ models are pushed into the production environment

From Software Dev CI/CD to MLOps + CT/CM

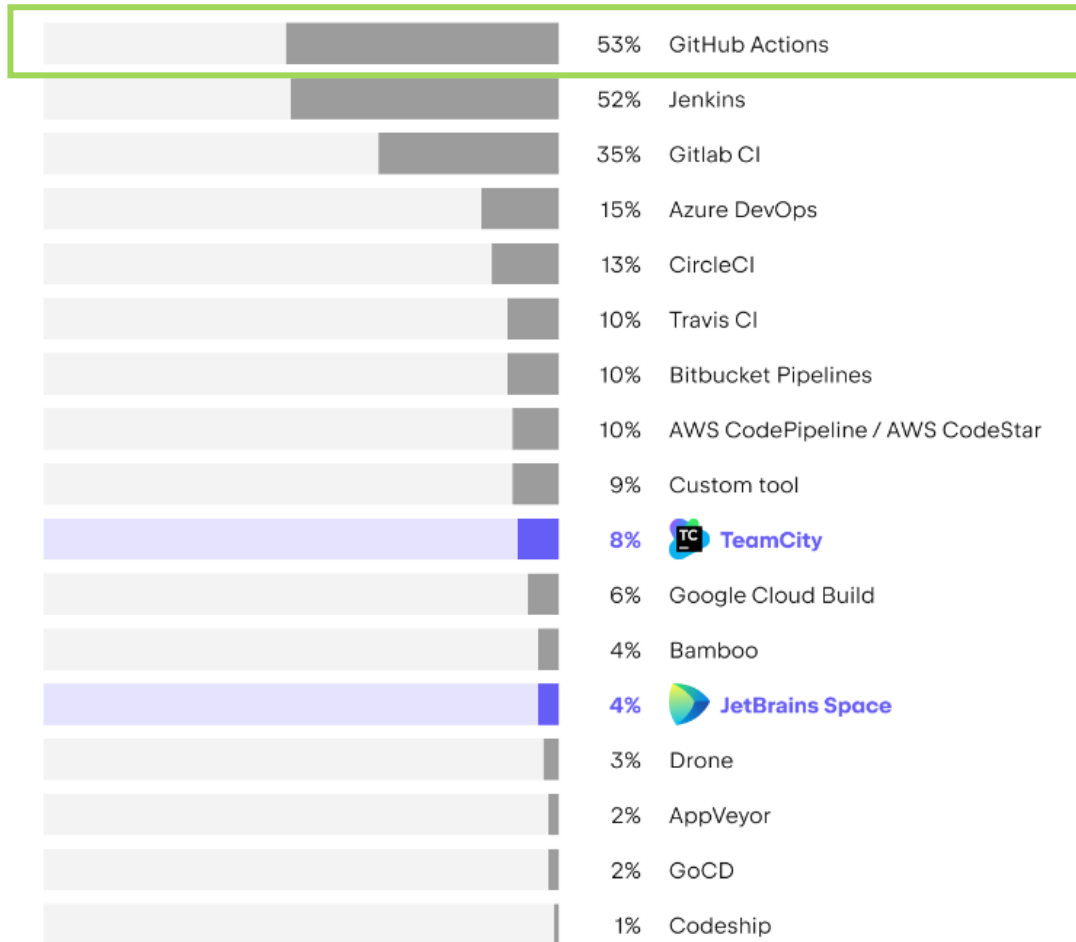
- **Continuous Integration (CI)** is not only about testing and validating code and components, but also **testing and validating data, data schemas, and models**
- **Continuous Delivery (CD)** is not only about a single software package or a service, but a system (an ML training pipeline) that **should automatically deploy another service** (model prediction service)
- **Continuous Training (CT)** is a new property, unique to ML systems, that's concerned with **automatically retraining candidate models** for testing and serving
- **Continuous Monitoring (CM)** is not only about catching errors in production systems, but also about **monitoring production inference data and model performance metrics** tied to business outcomes

Can be
trigger w/
GitHub
Actions

Using WandB

CI/CD in MLOps (via Github Actions) accelerates release frequency, enhancing user value and feedback collection

Which Continuous Integration (CI) systems do you regularly use?



Source: Jet Brains

- 1. Automated Workflows:** Streamlines CI/CD processes for machine learning models efficiently
- 2. Easy Integration:** Seamlessly integrates with existing GitHub repositories and tools
- 3. Customizable Pipelines:** Offers flexibility in designing tailored MLOps workflows
- 4. Community-Driven Actions:** Access to a vast repository of pre-built actions for varied tasks
- 5. Enhanced Security:** Provides built-in security features for secure MLOps operations.

Integrating GitHub Actions with MLflow & WandB automates workflows, ensures model consistency, and streamlines artifact management

1. **CI/CD:** Automates ML model testing, building, and deployment with each **code change** via Actions
2. **Automated Experiment Tracking:** Actions **triggers** MLflow experiments **automatically** on new code pushes, ensuring consistent experiment logging
3. **Automated Model Testing and Validation:** **New ML models** are automatically tested and validated in the Actions workflow
4. **Version Control for Models and Artifacts:** **Ensures** every model and artifact version is controlled and recorded
5. **Workflow Automation for Data Preparation and Processing:** **Triggers** data preparation and processing workflows, ensuring MLflow has **ready-to-use data** for model training
6. **Notifications and Reporting:** Set up Actions for proactive **notifications and reports** on ML experiment and model deployment statuses



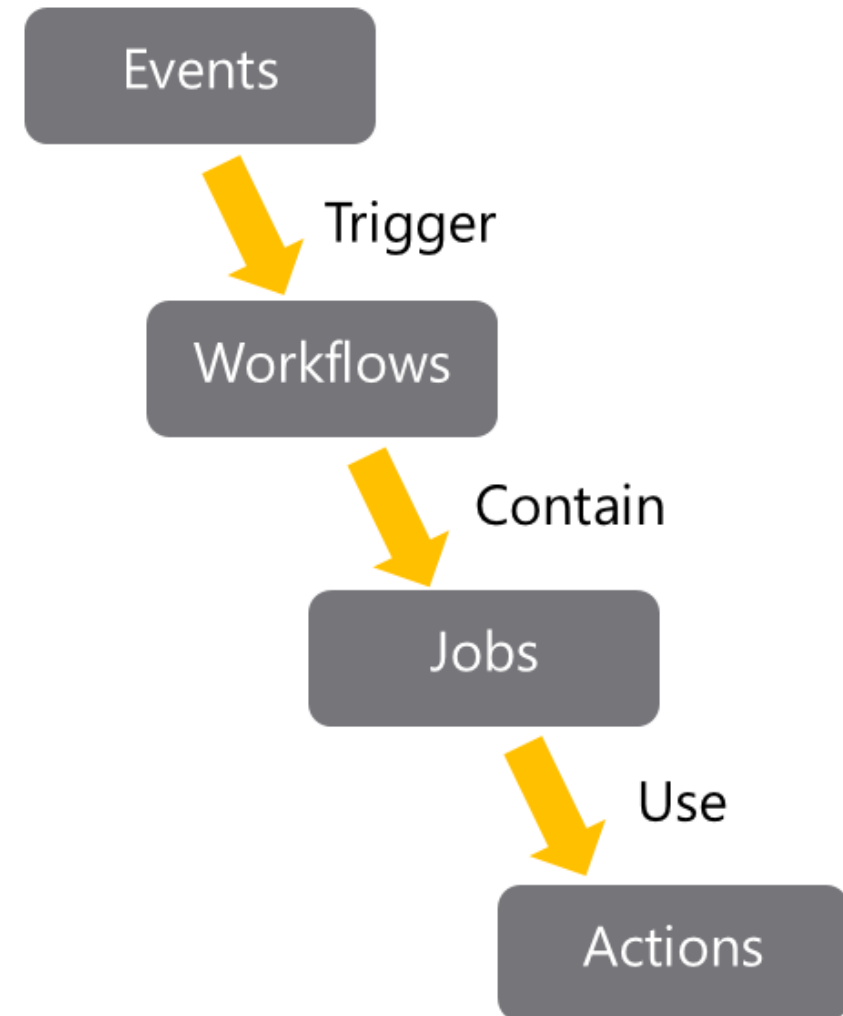
GitHub Actions



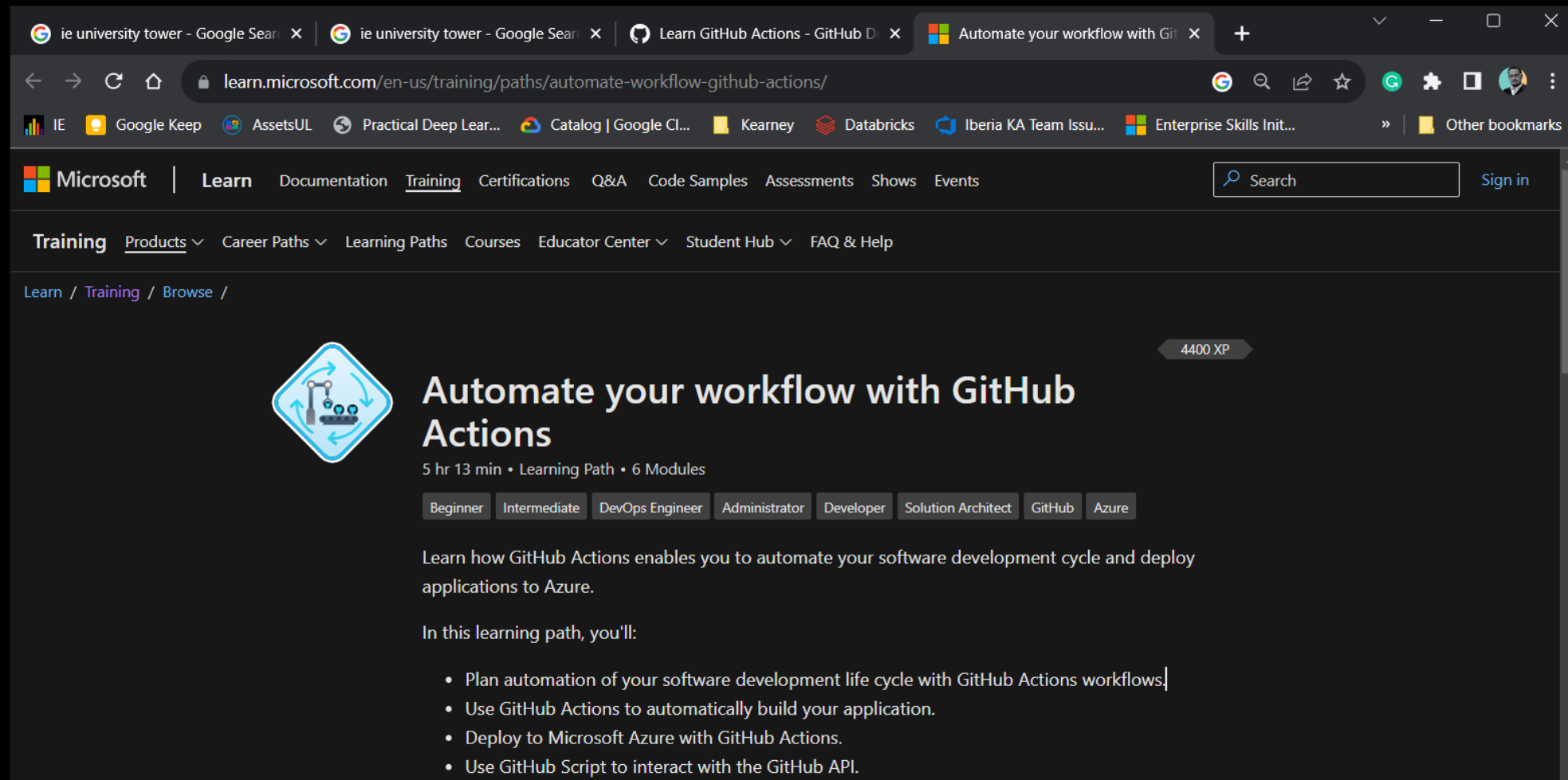
GitHub ACTIONS can do more than CI/CD !

- Automated testing
- Automatically responding to new issues, mentions
- Triggering code reviews
- Handling pull requests
- Branch management

Actions are executed on "runners," either hosted by GitHub or self-hosted



Optional: Introduction to GitHub Actions



The screenshot shows a web browser window with multiple tabs. The active tab is 'Learn GitHub Actions - GitHub D...'. The address bar shows the URL 'learn.microsoft.com/en-us/training/paths/automate-workflow-github-actions/'. The page header includes the Microsoft logo and navigation links: Learn, Documentation, Training (selected), Certifications, Q&A, Code Samples, Assessments, Shows, and Events. A search bar and a 'Sign in' link are also present. Below the header, the 'Training' section is expanded, showing 'Products', 'Career Paths', 'Learning Paths', 'Courses', 'Educator Center', 'Student Hub', and 'FAQ & Help'. The main content area features a blue diamond icon with a robot and arrows, representing automation. The title 'Automate your workflow with GitHub Actions' is prominently displayed, followed by '5 hr 13 min • Learning Path • 6 Modules'. A row of tags includes 'Beginner', 'Intermediate', 'DevOps Engineer', 'Administrator', 'Developer', 'Solution Architect', 'GitHub', and 'Azure'. A description states: 'Learn how GitHub Actions enables you to automate your software development cycle and deploy applications to Azure.' Below this, it says 'In this learning path, you'll:' followed by a bulleted list of objectives.


ie university tower - Google Search x | ie university tower - Google Search x | Learn GitHub Actions - GitHub D... x | Automate your workflow with Git x +

learn.microsoft.com/en-us/training/paths/automate-workflow-github-actions/

Microsoft | Learn Documentation Training Certifications Q&A Code Samples Assessments Shows Events Search Sign in

Training Products Career Paths Learning Paths Courses Educator Center Student Hub FAQ & Help

Learn / Training / Browse /

 **Automate your workflow with GitHub Actions** 4400 XP

5 hr 13 min • Learning Path • 6 Modules

Beginner Intermediate DevOps Engineer Administrator Developer Solution Architect GitHub Azure

Learn how GitHub Actions enables you to automate your software development cycle and deploy applications to Azure.

In this learning path, you'll:

- Plan automation of your software development life cycle with GitHub Actions workflows.
- Use GitHub Actions to automatically build your application.
- Deploy to Microsoft Azure with GitHub Actions.
- Use GitHub Script to interact with the GitHub API.

1 Start with pure CI (pytest only)

Running unit tests on every push gives immediate feedback and locks basic quality

Tasks

1. Add .github/workflows/ci.yml
2. Install deps, cache pip to speed repeats
3. Run pytest -q for Python 3.10 (or matrix e.g. 3.8-3.11)

```
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ['3.10', '3.11']
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: ${ matrix.python-version }
      - name: Cache pip
        uses: actions/cache@v3
        with:
          path: ~/.cache/pip
          key: ${ hashFiles('requirements*.txt') }
      - name: Install dependencies
        run: pip install -r requirements.txt -r
requirements-dev.txt
      - name: Run tests
        run: pytest -q
```

Canonical directory and updated files (illustrative)

```
my_project/
├── .github/
│   └── workflows/
│       └── ci.yml
├── tests/
│   └── test_api.py
├── environment.yml
├── requirements.txt
├── requirements-dev.txt
└── README.md
```


Anatomy of .github/workflows/ci.yml

```
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ['3.10', '3.11']
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: ${ matrix.python-version }
      - name: Cache pip
        uses: actions/cache@v3
        with:
          path: ~/.cache/pip
          key: ${ hashFiles('requirements*.txt') }
      - name: Install dependencies
        run: pip install -r requirements.txt -r
requirements-dev.txt
      - name: Run tests
        run: pytest -q
```

Anatomy of .github/workflows/ci.yml

```
name: CI
```

```
on:
```

```
  push:
```

```
    branches: [main]
```

```
  pull_request:
```

```
    branches: [main]
```

```
jobs:
```

```
  test:
```

```
    runs-on: ubuntu-latest
```

```
    strategy:
```

```
      matrix:
```

```
        python-version: ['3.10', '3.11']
```

```
    steps:
```

```
      - uses: actions/checkout@v4
```

```
      - name: Set up Python
```

```
        uses: actions/setup-python@v5
```

```
        with:
```

```
          python-version: ${ matrix.python-version }
```

```
      - name: Cache pip
```

```
        uses: actions/cache@v3
```

```
        with:
```

```
          path: ~/.cache/pip
```

```
          key: ${ hashFiles('requirements*.txt') }
```

```
      - name: Install dependencies
```

```
        run: pip install -r requirements.txt -r  
requirements-dev.txt
```

```
      - name: Run tests
```

```
        run: pytest -q
```

Triggers ("on:") Runs the workflow on every push and every pull request (PR) to the main branch.

Purpose: Ensures tests always run before code is merged to your main development branch

Job definition: Defines a job e.g. test

runs-on: Specifies the virtual machine OS e.g. Ubuntu (most common for Python CI)

Matrix strategy: Runs the same set of steps in parallel for different Python versions

Purpose: Catches compatibility issues with multiple Python versions (or any other SW)

Steps: Defines each action/ command run in the job

Checkout code: Downloads the repo to the CI environment so it can be tested

Install dependencies: Installs all runtime (**requirements.txt**) and dev/test (**requirements-dev.txt**)

Purpose: Ensures the test environment matches the local dev setup

Run tests: Executes pytest (or any other command)

Purpose: Fails the job if any test fails

Set up Python interpreter: Installs and sets the active Python version for the job

Purpose: Ensures correct Python version for each run

Cache dependencies: Uses the actions/cache to store (and later retrieve) downloaded Python packages

path: specifies what to cache i.e. ~/.cache/pip is pip's default download cache

key: Unique ID that updates whenever 'requirements.txt' change

Purpose: Speeds up repeated workflow runs

Workflow runs · 2025-IE-MLOps-course / mlops_project-CICD/

On your GitHub repo, go to **"Actions"** and you will see your CI job

Workflow run deleted successfully.

Actions

New workflow

All workflows

CI

Management

- Caches
- Attestations
- Runners
- Usage metrics
- Performance metrics

All workflows

Showing runs from all workflows

Filter workflow runs

2 workflow runs

	Event	Status	Branch	Actor
✓ chore: remove awscli from requirements.txt			main	
CI #4: Commit d84a3b4 pushed by phidesigner				
✗ chore: update Python version matrix in CI workflow			main	
Commit f991ce6 pushed by phidesigner				

Click on the job for further details, if a job (e.g. CI) fails, it won't let you merge to your branch



All checks have passed

2 successful checks



CI / test (3.10) (pull_request) Successful in 1m



CI / test (3.11) (pull_request) Successful in 1m



No conflicts with base branch

Merging can be performed automatically.

Merge pull request



You can also merge this with the command line. [View command line instructions.](#)

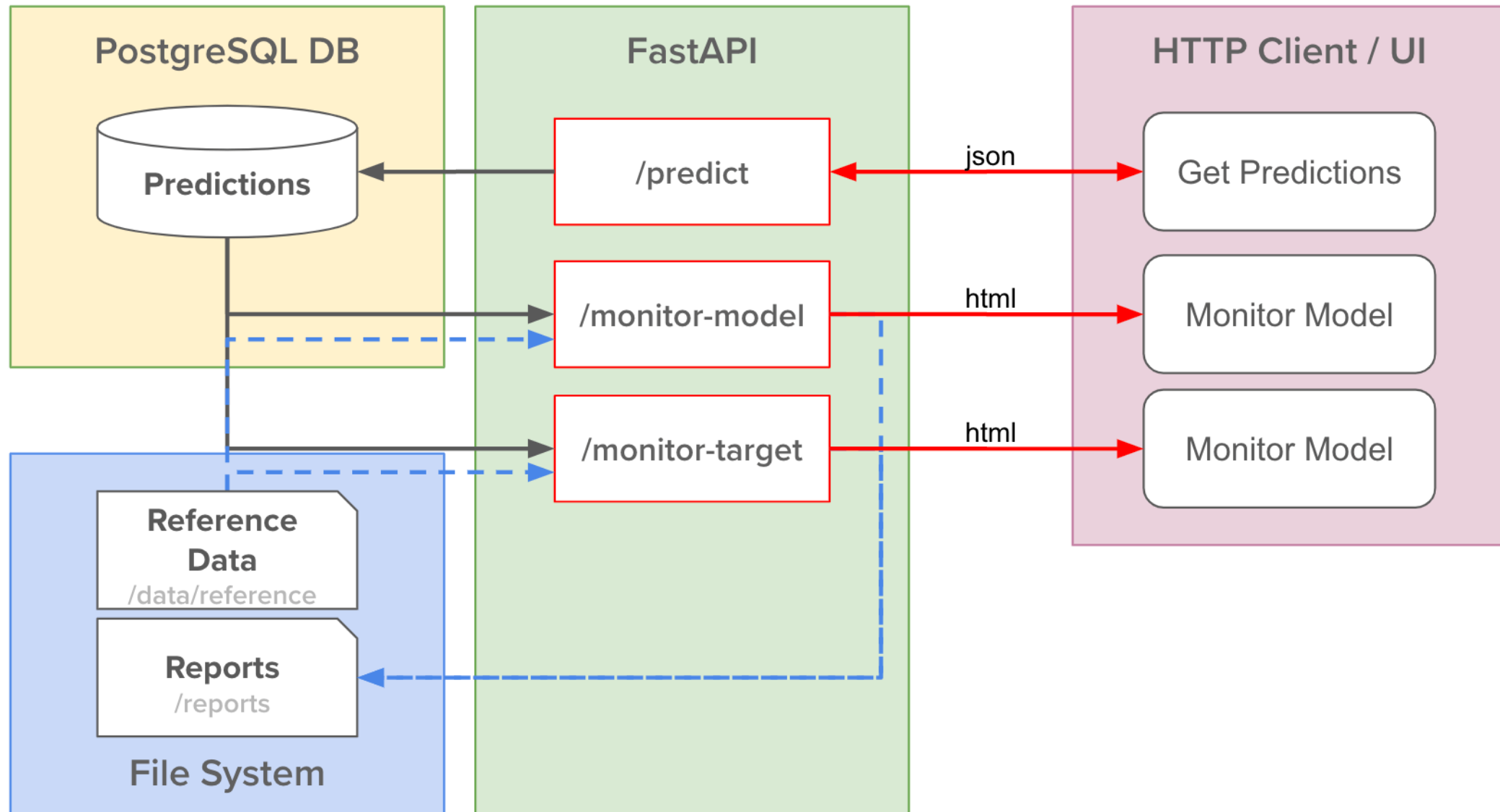
Exposing your learner with a high-performance, easy-to-use API using FastAPI

FastAPI for MLOps

- **FastAPI** is a tool to let apps talk to your ML model using standard web methods (verbs)
- **RESTful API** means using simple web commands (GET, POST...) to manage model services
- **HTTP endpoint** is the web address where you “ask” the model questions
- **Pydantic** checks and converts data to correct format automatically
- **OpenAPI docs** give interactive interface, so users test model calls in browser
- **Async I/O (Starlette + Uvicorn)** lets many users talk to the model at once
- REST API uses HTTP methods (verbs) to manage resources:
 - **GET** → Ask model a question, e.g. “What’s today’s temperature?”
 - **POST** → Send data to model, e.g. “Analyse this photo”
 - **PUT/PATCH** → Update model settings, e.g. “Change performance metric”
 - **DELETE** → Remove a model or its data, e.g. “Erase old version”

Typical FastAPI ML flow: import FastAPI > define endpoints > load model > handle predict route

Simple architecture schema



Source: <https://medium.com/data-science/step-by-step-approach-to-build-your-machine-learning-api-using-fast-api-21bd32f2bbdb>

2

Add FastAPI service

Exposing a lightweight API before containerising and deploy on a remote server

Tasks

1. Create app/main.py with FastAPI()
2. Include /health returning {"status": "ok"} for probes
3. Add requirements-api.txt with fastapi uvicorn

```
my_project/
├── app/
│   ├── __init__.py
│   └── main.py
├── tests/
│   └── test_api.py
├── scripts/
│   └── call_api.py
├── .env
├── pytest.ini
├── environment.yml
├── requirements.txt
├── requirements-dev.txt
└── README.md
```


Anatomy of */app/main.py*

```
(...)
RAW_FEATURES = CONFIG.get("raw_features",
[])
ENGINEERED = CONFIG.get("features",
{}).get("engineered", [])

app = FastAPI()

class PredictionInput(BaseModel):
    rx_ds: int = Field(..., alias="rx ds")
    A: int
    B: int
    C: int
    ...
    class Config:
        allow_population_by_field_name =
True
        schema_extra = {
            "example": {
                "rx_ds": 100,
                "A": 0,
                "B": 1,
                "C": 0,
                ... }}
(...)

```

```
(...)

@app.get("/")
def root():
    return {"message": "Welcome to the
opioid abuse prediction API"}

@app.get("/health")
def health():
    return {"status": "ok"}

@app.post("/predict")
def predict(payload: PredictionInput):
    data = payload.dict(by_alias=True)
    df = pd.DataFrame([data])
    X_raw = df[RAW_FEATURES]
    X_proc = PIPELINE.transform(X_raw)
    if ENGINEERED:
        feature_names =
get_output_feature_names(
            preprocessor=PIPELINE,
            input_features=RAW_FEATURES,
            config=CONFIG,
        )
        indices = [feature_names.index(f)
for f in ENGINEERED if f in feature_names]
        X_proc = X_proc[:, indices]
        pred = MODEL.predict(X_proc)[0]
        proba = MODEL.predict_proba(X_proc)[0,
1] if hasattr(MODEL, "predict_proba") else
None
        return {"prediction": int(pred),
"probability": float(proba) if proba is
not None else None}

```

Initialize our **FastAPI**

```
(...)  
app = FastAPI()
```

Defines the input data schema using **Pydantic**

- Each feature as a type-checked field
- `rx_ds` is mapped to/from "`rx ds`" for alias support
- `Class Config` enables field aliases and provides an example payload for docs

```
class PredictionInput(BaseModel):  
    rx_ds: int = Field(..., alias="rx ds")  
    A: int  
    B: int  
    C: int  
    ...  
    class Config:  
        allow_population_by_field_name =  
            True  
        schema_extra = {  
            "example": {  
                "rx_ds": 100,  
                "A": 0,  
                "B": 1,  
                "C": 0,  
                ... }  
            }  
        (...)
```

Prediction endpoint:

- Accepts a POST request with validated input data
- Converts input to a DataFrame, applies the preprocessing pipeline
- Optionally selects engineered features
- Uses the loaded ML model to predict the outcome and probability
- Returns prediction and probability as JSON.

```
(...)
```

Root endpoint with simple message

```
@app.get("/")  
def root():  
    return {"message": "Welcome to the  
opioid abuse prediction API"}
```

Check endpoint

```
@app.get("/health")  
def health():  
    return {"status": "ok"}
```

```
@app.post("/predict")  
def predict(payload: PredictionInput):  
    data = payload.dict(by_alias=True)  
    df = pd.DataFrame([data])  
    X_raw = df[RAW_FEATURES]  
    X_proc = PIPELINE.transform(X_raw)  
    if ENGINEERED:  
        feature_names =  
            get_output_feature_names(  
                preprocessor=PIPELINE,  
                input_features=RAW_FEATURES,  
                config=CONFIG,  
            )  
        indices = [feature_names.index(f)  
for f in ENGINEERED if f in feature_names]  
        X_proc = X_proc[:, indices]  
        pred = MODEL.predict(X_proc)[0]  
        proba = MODEL.predict_proba(X_proc)[0,  
1] if hasattr(MODEL, "predict_proba") else  
None  
        return {"prediction": int(pred),  
"probability": float(proba) if proba is  
not None else None}
```

Anatomy of */app/main.py*

Run, test, and
validate your
FastAPI
prediction
service **locally**
using scripts,
browser
endpoints, and
interactive docs

Running your app locally

```
> uvicorn app.main:app --reload
```

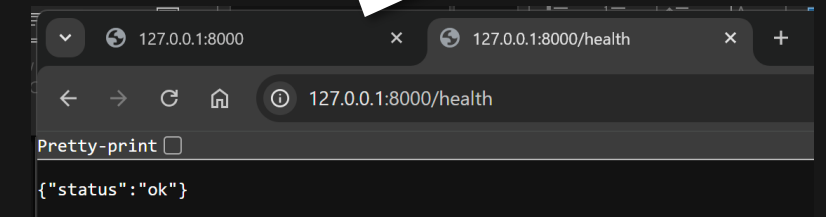
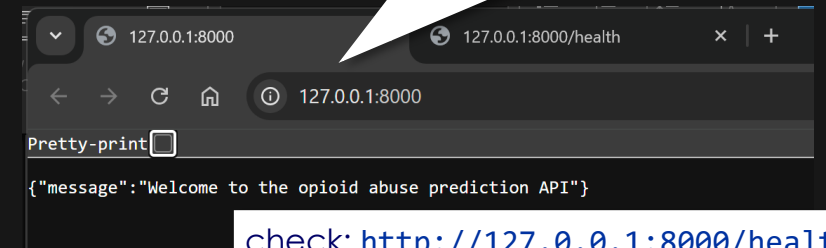
Test via script

```
> python scripts/call_api.py --url  
http://127.0.0.1:8000/predict --input  
/home/idiiazl/2025_MLOps/mlops_project-  
CICD/data/inference/new_data.csv
```

```
> python scripts/call_api.py --url  
http://127.0.0.1:8000/predict_batch --  
input  
/home/idiiazl/2025_MLOps/mlops_project-  
CICD/data/inference/new_data.csv
```

- check: <http://127.0.0.1:8000/docs>
- Interactive docs at `/docs` to try your `/predict` endpoint with sample data

- Starts your API on <http://127.0.0.1:8000>
- `--reload` auto-restarts when you change code



default

GET / Root

GET /health Health

POST /predict Predict

Parameters

No parameters

Request body required

application/json

Edit Value | Schema

Paste your data for inference `/predict` entry point

Execute

Clear

http://127.0.0.1:8000/ - My Workspaces

web.postman.co/workspace/My-Workspace~7e039b2c-f405-4ef1-86d5-688bac6ac91b/request/create?requestId=97480...

Home Workspaces API Network

GET http://127.0.0.1:8000/

You can alternatively check your API with postman

GET http://127.0.0.1:8000/

Params Authorization Headers (7) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (4) Test Results

{ } JSON Preview Visualization

```
1 {
2   "message": "Welcome to the opioid abuse prediction
3 }
```

POST http://127.0.0.1:8000/predict

Params Authorization Headers (10) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "rx ds": 0,
3   "A": 0,
4   "B": 0,
5   "C": 0,
6   "D": 0,
7   "F": 0.

```

Body Cookies Headers (4) Test Results

{ } JSON Preview Visualization

```
1 {
2   "prediction": 0,
3   "probability": 0.0
4 }
```

200 OK • 20 ms • 159 B

Paste your data for inference */predict*

3

Dockerise the project

The Docker image isolates runtime and eases deployment to remote servers e.g. Render, Railway, etc.

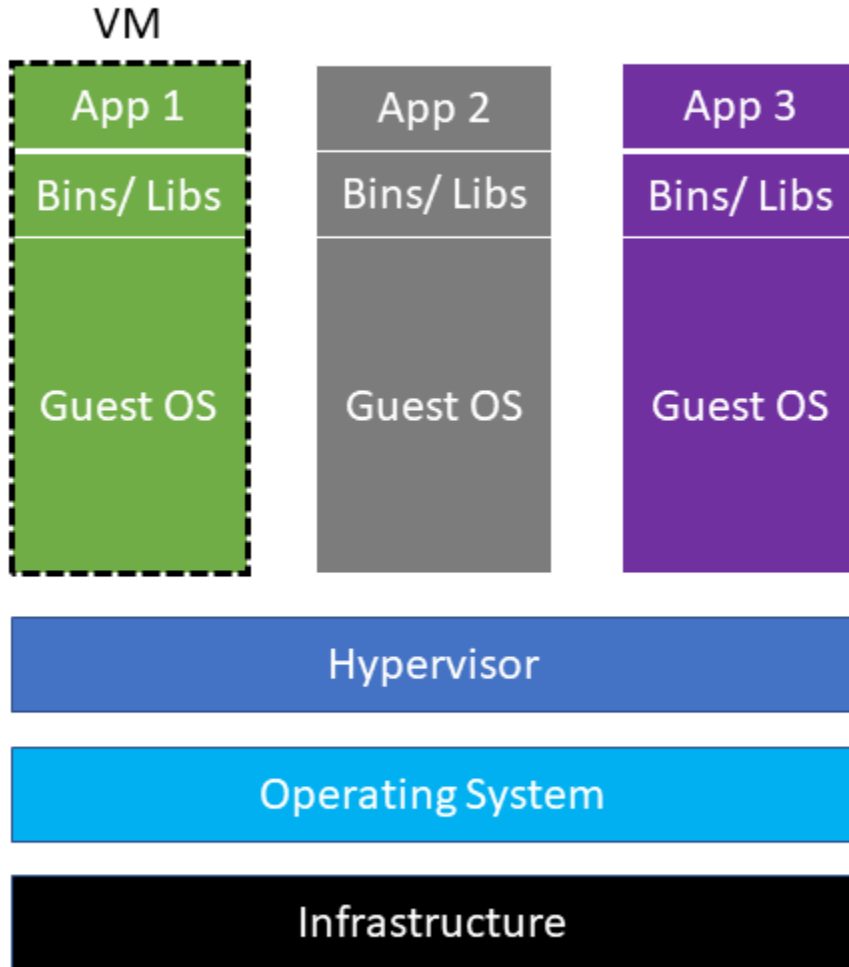
Tasks

1. Install Docker desktop
2. Create Dockerfile
3. .dockerignore
4. Configure render.yml
5. Add requirements (-api).txt
 1. /app
 2. /inference
 3. config, .env, scripts and models

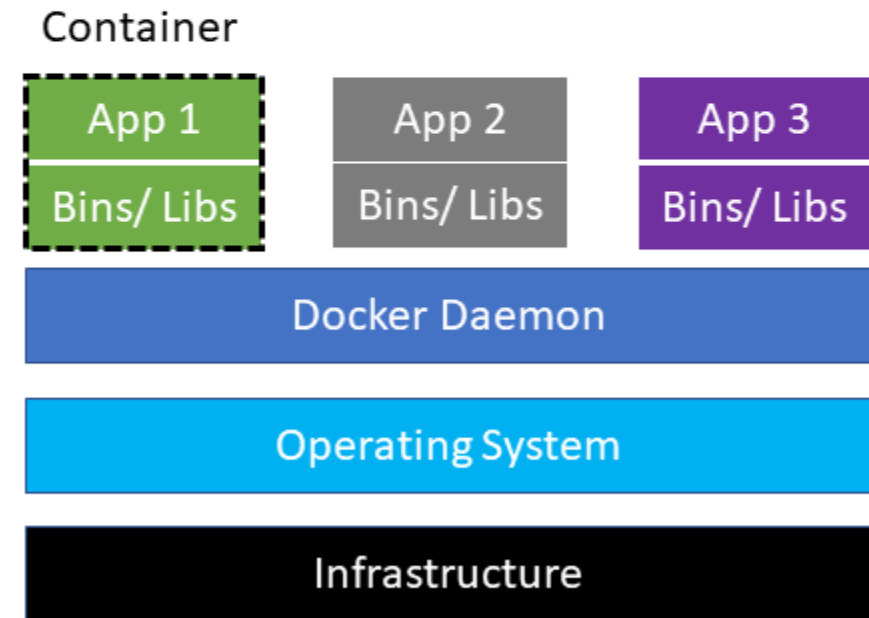
```
my_project/
├── app/
│   ├── __init__.py
│   └── main.py
├── Dockerfile
├── .dockerignore
├── render.yml
├── requirements.txt
├── requirements-dev.txt
└── README.md
```

Different to VMS, Docker images don't run on their own OS

Virtual Machines vs. Containers



Virtual Machines



Docker Containers

Anatomy of /Dockerfile

- Sets **working directory** inside the container to `/code`, so all following commands run from there
- **ENV** tells Python where to find your custom modules in `/code/src`

Copies dependency files from the local machine into the container (Dev and Prod)

- Installs all Python dependencies
- `--no-cache-dir` keeps the image size smaller by not saving pip's download cache

Defines a simple check to see if the app is healthy. If it fails, marks the container as unhealthy

- Runs a script to download artifacts from Weights & Biases
- Launches the FastAPI app with Uvicorn on port 8000 (or default)

```
FROM python:3.10-slim
```

Uses a lightweight **Python** image as base, to minimize image size

```
WORKDIR /code  
ENV PYTHONPATH=/code/src
```

```
# install dependencies first to leverage  
layer caching  
COPY requirements.txt requirements-dev.txt  
./  
RUN pip install --no-cache-dir -r  
requirements.txt -r requirements-dev.txt
```

```
# copy application code  
COPY app ./app  
COPY src ./src  
COPY config.yaml ./  
COPY main.py ./  
COPY MLproject ./  
COPY scripts ./scripts
```

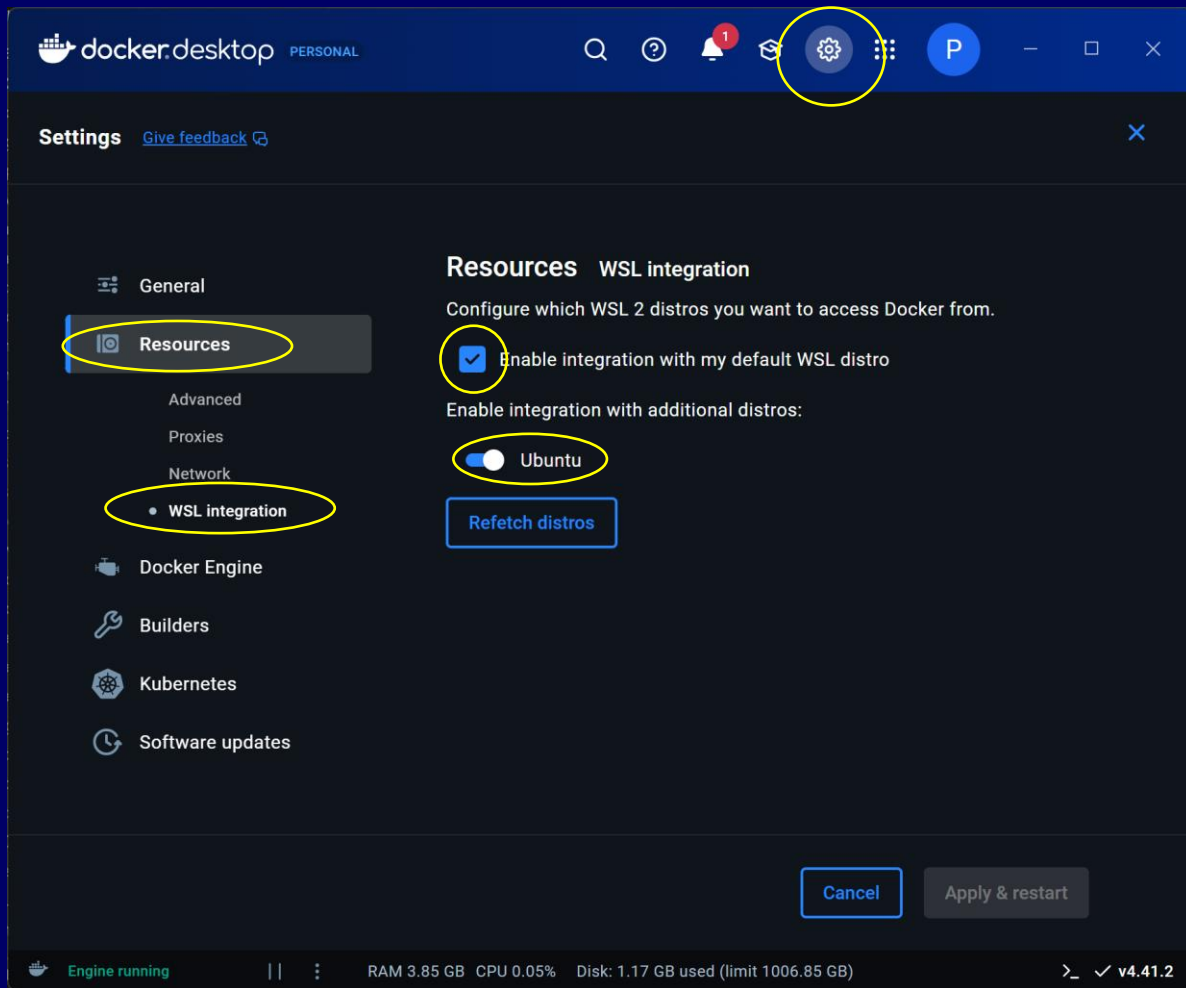
Copies the app code and configs into the image. Ensuring all required source files and configs are available for the app to run (together with `.dockerignore`)

```
EXPOSE 8000
```

```
HEALTHCHECK CMD curl --fail  
http://localhost:${PORT:-8000}/health ||  
exit 1
```

```
CMD ["sh", "-c", "python  
scripts/download_from_wandb.py && uvicorn  
app.main:app --host 0.0.0.0 --port ${PORT:-  
8000}"]
```

For WSL users make sure WSL is integrated



Docker set up checks

```
> docker version
> Docker run hello-world
```

1. Set/ update WANDB credentials on your .env

```
WANDB_PROJECT=opioid_mlops_project_CICD
WANDB_ENTITY=idiazl
WANDB_API_KEY=xxxx
```

2. Build the image

```
> docker build -t <opioid-api> .
```

Check for created images

```
> docker images
```

3. Run the container

```
> docker run --env-file .env -p 8000:8000 opioid-api
```

4. Check status via app, docs, or terminal

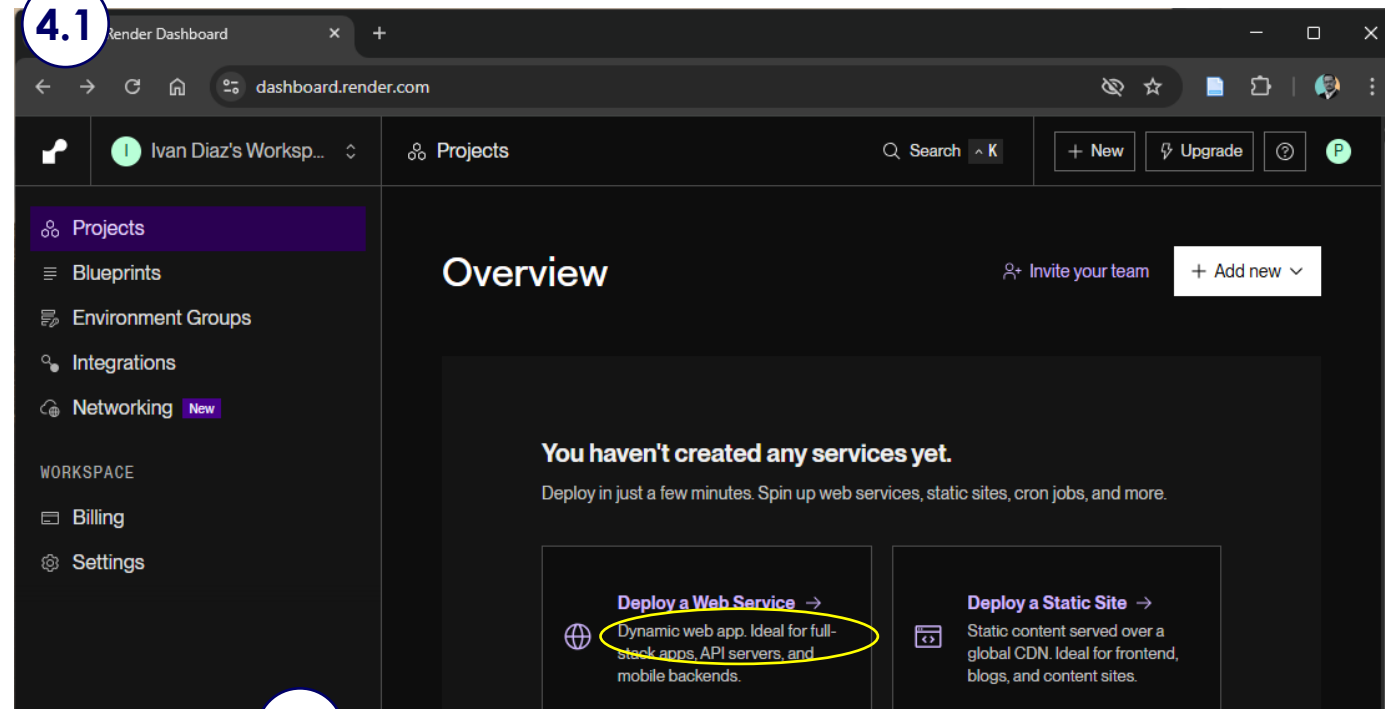
```
> curl http://localhost:8000/health
```

5. Submit a local testing batch (via script)

```
> python scripts/call_api.py --url
http://localhost:8000/predict_batch --input
data/inference/new_data.csv
```

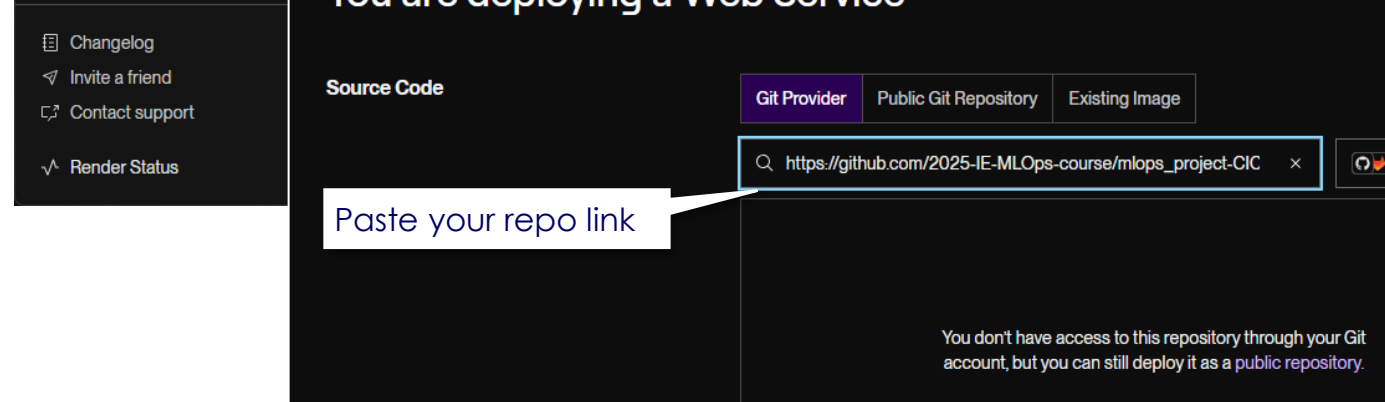
4 Deploying on Render.com (remote)

4.1



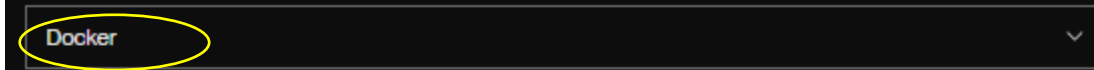
4.2

You are deploying a Web Service



4.3

Language



Branch

The Git branch to build and deploy.



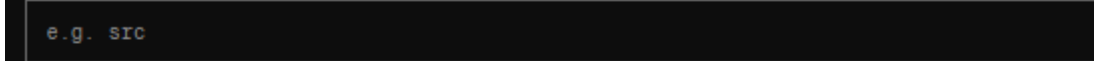
Region

Your services in the same region can communicate over a private network.



Root Directory Optional

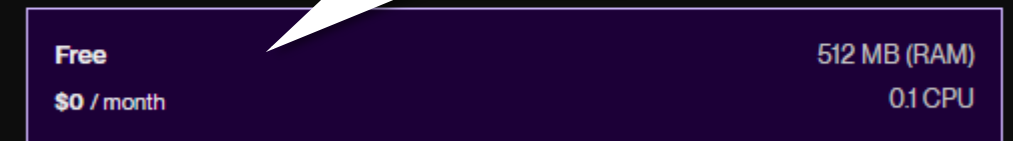
If set, Render runs commands from this directory instead of the repository root. Additionally, code changes outside of this directory do not trigger an auto-deploy. Most commonly used with a monorepo.



Instance Type

For hobby projects

Select Free option



⚠ Upgrade to enable more features

Free instances spin down after periods of inactivity. They do not support SSH access, scaling, one-off jobs, or persistent disks. Select any paid instance type to enable these features.

Environment Variables

Set environment-specific config and secrets. You can set those values from your code. [Learn more.](#)

WANDB_PROJECT

WANDB_ENTITY

WANDB_API_KEY

+ Add Environment Variable

Add from .env

Set or upload your wandb credentials

WEB SERVICE

mlops_project-CICD

Docker

Free

Upgrade your instance →

Connect

Manual Deploy

2025-IE-MLOps-course / mlops_project-CICD

<https://mlops-project-cicd.onrender.com>

You'll be notified
once your app is live

ⓘ Your free instance will spin down with inactivity, which can delay restarts by 50 seconds or more.

Upgrade now

June 13, 2025 at 5:26 PM

Live

3d8e922 Update decision tree model binary file

All logs

Search

Live tail

GMT+2

```
Jun 13 05:32:06 PM wandb: Tracking run with wandb version 0.19.11
Jun 13 05:32:06 PM wandb: Run data is saved locally in /code/wandb/run-20250613_153205-fs55bioa
Jun 13 05:32:06 PM wandb: Run 'wandb offline' to turn off syncing.
Jun 13 05:32:06 PM wandb: Syncing run eager-mountain-17
Jun 13 05:32:06 PM wandb: ⭐ View project at https://wandb.ai/idiarz/opioid_mlops_project_CICD
Jun 13 05:32:06 PM wandb: 🔥 View run at https://wandb.ai/idiarz/opioid_mlops_project_CICD/runs/fs55bioa
Jun 13 05:32:09 PM wandb: \ 1 of 4 files downloaded...wandb: 4 of 4 files downloaded.
Jun 13 05:32:10 PM wandb: 1 of 1 files downloaded.
Jun 13 05:32:11 PM wandb:
Jun 13 05:32:11 PM wandb: 🔥 View run eager-mountain-17 at: https://wandb.ai/idiarz/opioid_mlops_project_CICD/runs/fs55bioa
Jun 13 05:32:11 PM wandb: ⭐ View project at: https://wandb.ai/idiarz/opioid_mlops_project_CICD
Jun 13 05:32:11 PM wandb: Synced 5 W&B file(s), 0 media file(s), 0 artifact file(s) and 0 other file(s)
Jun 13 05:32:11 PM wandb: Find logs at: ./wandb/run-20250613_153205-fs55bioa/logs
Jun 13 05:32:24 PM /usr/local/lib/python3.10/site-packages/pydantic/_internal/_config.py:373: UserWarning: Valid config keys have changed in V2:
Jun 13 05:32:24 PM * 'allow_population_by_field_name' has been renamed to 'populate_by_name'
Jun 13 05:32:24 PM * 'schema_extra' has been renamed to 'json_schema_extra'
Jun 13 05:32:24 PM ⚠ warnings.warn(message, UserWarning)
Jun 13 05:32:24 PM INFO: Started server process [62]
Jun 13 05:32:24 PM INFO: Waiting for application startup.
Jun 13 05:32:24 PM INFO: Application startup complete.
Jun 13 05:32:24 PM INFO: Uvicorn running on http://0.0.0.0:1936 (Press CTRL+C to quit)
Jun 13 05:32:25 PM INFO: 127.0.0.1:40200 - "HEAD / HTTP/1.1" 405 Method Not Allowed
Jun 13 05:32:32 PM ==> Your service is live 🚀
```

You'll be notified
once your app is live

Testing the remote app programmatically

```
python scripts/call_api.py --url
https://mlops-project-
cicd.onrender.com/predict_batch -
-input
data/inference/new_data.csv
```