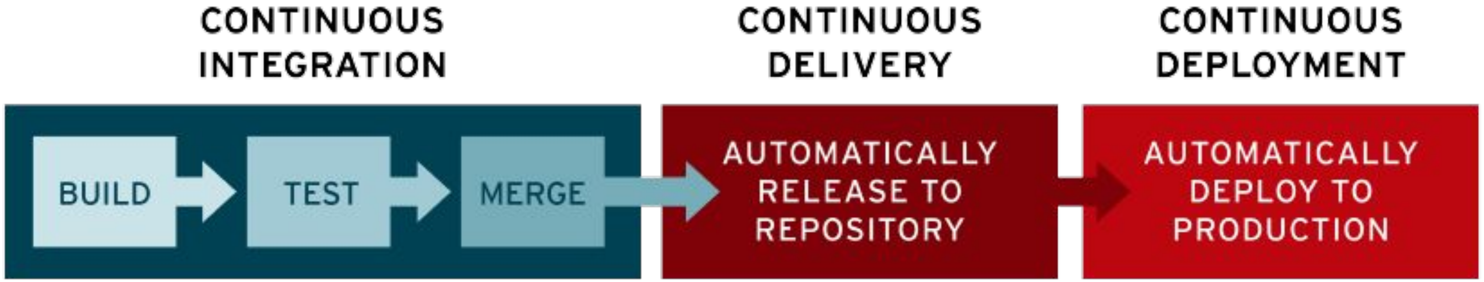
CI/CD

modulai.io

modulai

CI/CD

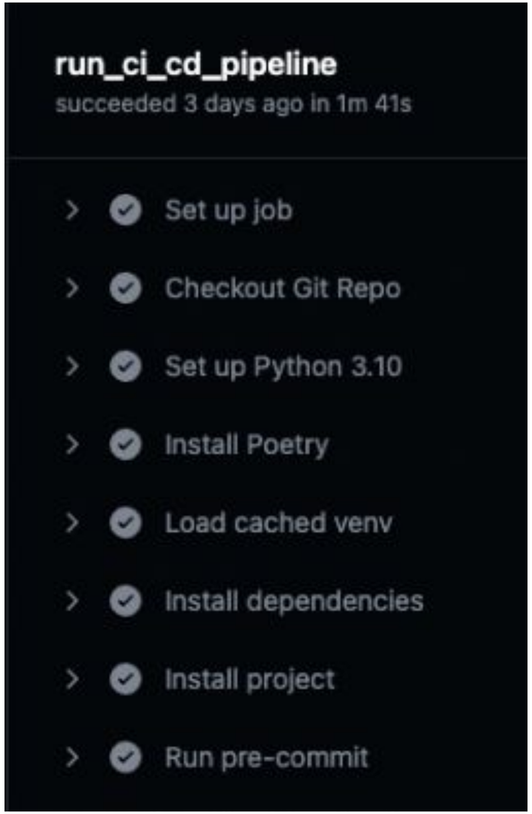
- Automatically run pipelines on new code changes to check for issues or bugs.
- Shared rules which lead to better code quality.
- Fast feedback loop for developers.
- Promotes transparency and knowledge sharing.



modulai

CI/CD in Practice

1. Develop code changes.
2. Push code to Github.
3. Github automatically detect new code and starts CI
4. CI downloads code, installs virtual environment, runs tests, etc...



run_ci_cd_pipeline
succeeded 3 days ago in 1m 41s

- ✓ Set up job
- ✓ Checkout Git Repo
- ✓ Set up Python 3.10
- ✓ Install Poetry
- ✓ Load cached venv
- ✓ Install dependencies
- ✓ Install project
- ✓ Run pre-commit

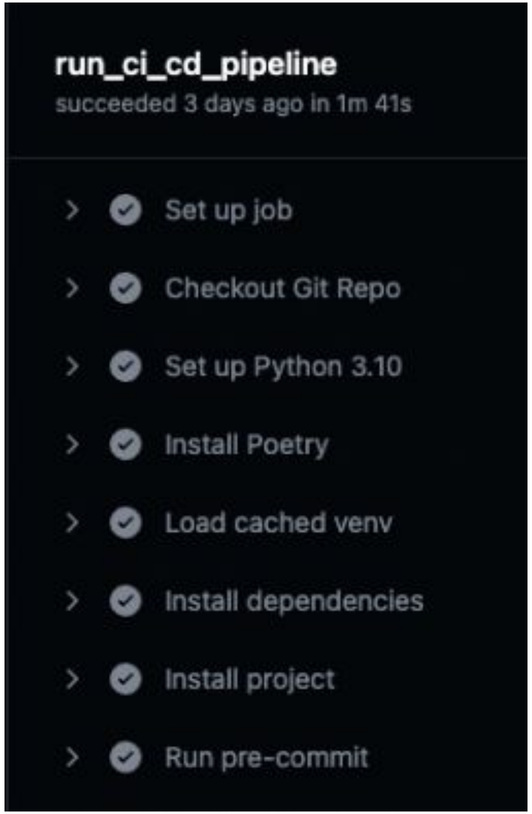
CI/CD in Practice

CI/CD can also do other things.

- Train and test the ML model.
- Create reports and share them with stakeholders.
- Ask chatGPT for code improvements .

What *should* CI/CD do?

It depends on the project situation. CI will run often, so you don’t want it to be expensive (OpenAI API) or run very slowly (train a big ML model).



run_ci_cd_pipeline
succeeded 3 days ago in 1m 41s

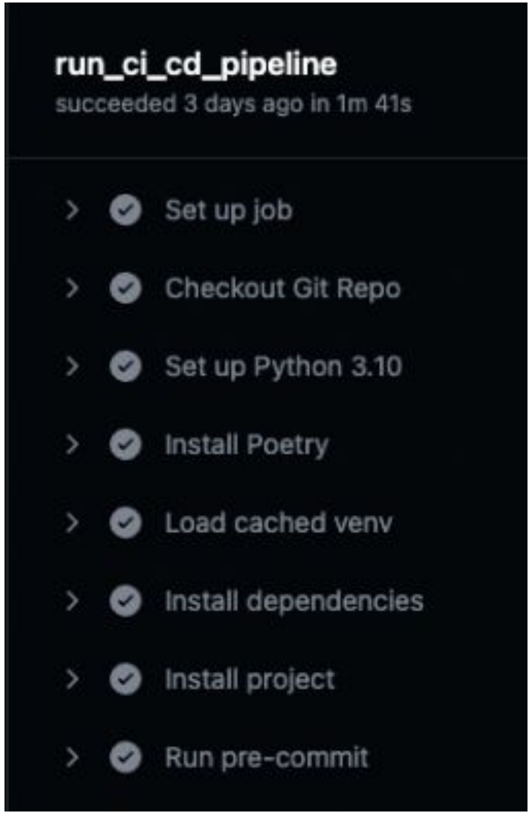
- Set up job
- Checkout Git Repo
- Set up Python 3.10
- Install Poetry
- Load cached venv
- Install dependencies
- Install project
- Run pre-commit

modulai

CI/CD in Practice

What will we do in the project.

- Download code from github.
- Install our python requirements into our venv.
- Check our code for style/format errors.
- Run tests.
- (Anything else you want to try for fun). 🤔



run_ci_cd_pipeline
succeeded 3 days ago in 1m 41s

- Set up job
- Checkout Git Repo
- Set up Python 3.10
- Install Poetry
- Load cached venv
- Install dependencies
- Install project
- Run pre-commit

modulai

CI/CD Config File

GitHub reads the config file

my-git-repo/.github/workflows/example.yaml

We can configure

- When the workflow runs. Every git push. Every sunday.
- What OS should be used (Mac, Windows, ...)
- What Python version we should use. We can even repeat the same workflow with different Python versions

```yaml
name: Python run CI CD
on: pull_request
jobs:
 run_ci_cd_pipeline:
   runs-on: ubuntu-latest
   steps:
   # Git clone repo
   - name: Checkout Git Repo
     uses: actions/checkout@v3

   # Install Python
   - name: Set up Python 3.10
     uses: actions/setup-python@v4
     with:
       python-version: '3.10'

   # Install requirements
   - name: Install dependencies
     run: |
       pip install -r requirements.txt

   # Run tests
   - name: Test with pytest
     run: |
       pytest tests/
```

Black: Code formatter

🙏Black: https://github.com/psf/black 🙏

Changes your code to follow the black “style”.

Only cosmetic changes. Code behaves 100% the same.

Easier to read code - avoids different styles in large teams

```
# python_example.py
a=1
b = [1,2
3]



c = 1*2 + 5
# Black formats the code 👇.
Space between operators, simplify, max 2
empty new-lines and many other things.

a = 1
b = [1,2,3]

c = 1 * 2 + 5
```

modulai

isort: Import organizer

isort: https://github.com/PyCQA/isort

(Not as nice as Black, but pretty good! 🤗)

Changes the order of your imports at the top of the file

1. Python "built-ins" e.g. time, random, pathlib, os, sys
2. Third party e.g. pandas, numpy, dash
3. Project specific imports. For files and functions you created yourself

```
import pandas as pd
from my_project import make_paper_clips
from pathlib import Path
import numpy as np
from my_project.matrix import subdue_human_race
import os

# isort formats the code 👇.
# 1. Python built-ins
import os
from pathlib import Path

# 2. Third party
import numpy as np
import pandas as pd

# 3. Our own code
from my_project import make_paper_clips
from my_project.matrix import subdue_human_race
```

modulai

Tests

Tests help identify and fix issues, leading to higher-quality software and reducing the risk of important errors.

Great for detecting unintended side-effects.

Describes how the code is intended to be used and provide examples of its expected behavior.

🤫Tests can be skipped in certain cases, such as small hobby projects with minimal consequences for errors, compared to high-stakes applications like self-driving cars.🤫

“The more you trust your tests, the more you're like, oh, I got a pull request and the tests pass, I feel okay to merge that, the faster you can make progress.”

George Hotz

(Famous hacker/programmer)

modulai

Tests: Different Types

- **Unit Tests**: Focus on verifying the smallest testable units of code, such as individual functions or methods.
- **Integration Tests**: Integration tests verify the interaction and integration between different modules or components.
- **Regression Tests**: Help identify any unintended side effects or regressions caused by the modifications.
- Acceptance Tests, Functional Tests, Performance Tests, Security Tests etc… What matters is that there are **different types of tests** - the **importance depends** on the project.

You want your tests to be fast. That allows them to run often without disturbing the developers productivity.

Pytest

Pytest: https://docs.pytest.org/

Most popular testing framework in Python.

Pytest can be used to create and run tests.

Pytest will automatically find your tests if you start their names with “test”.

```
# Put test code under tests/
├── src
|    └── newsfeed
|         ├── example1.py
|         └── example2.py
└── tests
    ├── test_example1.py
    └── test_example2.py
```

Pytest

```python
# tests/test_example1.py
import platform

# Pytest find this function cuz it starts with “test”
def test_correct_python_version():
    python_version = platform.python_version()
    assert python_version == "3.10.2"

# Pytest ignores this one
def helper_function_not_starting_with_test():
    return 😓
```

```
# Put test code under tests/
├── src
│    └── newsfeed
│         ├── example1.py
│         └── example2.py
└── tests
     ├── test_example1.py
     └── test_example2.py
```

modulai

Pytest advanced

```python
from typing import Generator

import pytest
from src.api.database import Credentials, get_cursor,
get_routes_from_db
from src.api.datamodel import Waste
from src.utils.datatypes import Error

@pytest.mark.order("first")
def test_connect_db() -> None:
    try:
        get_cursor(Credentials.routes())
    except Exception as err:
        pytest.exit(f"Couldn't connect to the database.")

def test_setup_db(setup_db: Generator) -> None:
    pass
```

```python
def test_stored_procedure(setup_db: Generator) -> None:
    route_db_cursor = get_cursor(Credentials.routes())
    waste = [Waste(waste_type="HEMSORT")]
    routes = get_routes_from_db(waste)
    assert not isinstance(routes, Error)
    for route in routes:
        if route.vehicle == "22":
            assert len(route.stops) == 3, route
        if route.vehicle == "21":
            assert False, route
        if route.date != "TOMORROW":
            assert False, route
```

Pre-commit

⭐Pre-commit: https://pre-commit.com/ ⭐

Mini CI/CD that runs on your laptop every time you make a git commit.

Save so much time since it’s much quicker than Github CI/CD

```yaml
# .pre-commit-config.yaml
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
 rev: v4.4.0
 hooks:
 - id: trailing-whitespace
 - id: check-yaml
 - id: check-toml
 - id: check-added-large-files
 - id: debug-statements
   language_version: python3
- repo: https://github.com/jumanjihouse/pre-commit-hook-yamlfmt
 rev: 0.2.3
 hooks:
 - id: yamlfmt
- repo: https://github.com/psf/black
 rev: 23.3.0
 hooks:
 - id: black
   language_version: python3
- repo: https://github.com/pycqa/isort
 rev: 5.12.0
 hooks:
 - id: isort
   name: isort (python)
```

Honorable Mentions

Flake8 https://flake8.pycqa.org/en/latest/

Can tell you if you have created variables that are never used and many more helpful code checks.

```python
import math

def calculate_circle_area(radius):
    diameter = radius * 2  # Flake8 warning: Variable not used
    area = radius * radius * math.pi
    return area
```

Ruff https://github.com/astral-sh/ruff

New tool that is growing very quickly. Looks like this will be the default tool soon - but the older tools have better compatibility right now :)

“Ruff can be used to replace Flake8, isort, pydocstyle, yesqa, eradicate, pyupgrade, and autoflake, all while executing tens or hundreds of times faster than any individual tool.”