Clase 197 — CI/CD para ML con GitHub Actions

Parte: 4 — MLOps · Fuente: Huyen cap. 10 + docs GitHub Actions + CML.dev. Duración estimada: 80 min.

Clase 197 — CI/CD para ML con GitHub Actions

> *Parte: 4 — MLOps · Fuente: Huyen cap. 10 + docs GitHub Actions + CML.dev. Duración estimada: 80 min.*

Objetivo

Automatizar el ciclo lint → test → entrenar → evaluar → comparar → desplegar con GitHub Actions. Cada PR dispara entrenamiento sobre el slice actual de datos, postea métricas como comentario, y bloquea merge si la métrica empeoró >X%. Sin CI/CD, "el modelo nuevo es mejor" es opinión; con CI/CD, es un workflow check o .

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Escribir un workflow .github/workflows/ml.yml con jobs lint, test, train, evaluate.
- Usar CML (Continuous Machine Learning) para postear plots/metrics en el comentario del PR.
- Cachear dependencias (actions/setup-python con cache: pip) y pesos de modelos (actions/cache) para que el CI no tarde 20 min.
- Usar secrets y OIDC para deploy seguro a AWS/GCP sin claves long-lived.
- Configurar branch protection + required checks que bloqueen merge si las métricas degradan.

Temas

#	Tema	Por qué importa
1	Anatomía de un workflow: triggers, jobs, s	Vocabulario base.
2	Matrix builds (strategy.matrix)	Probar 3 versiones de Python × 2 de PyTorc
3	Caché de pip + datasets pesados	CI <5 min vs CI >30 min.
4	CML: reportar métricas en PR	Trae al code review los números, no solo e
5	OIDC para deploy (sin secret long-lived)	Federation con AWS/GCP/Azure.
6	Branch protection + required status checks	Convierte el lint/test/eval en gate, no en

Definiciones y características

- Workflow: archivo YAML en .github/workflows/*.yml. Disparado por on: (push, pull_request, schedule, workflow_dispatch).
- Job: agrupación de steps que corren en el mismo runner (VM/container). Jobs corren en paralelo por default; usar needs: [job1] para serializar.
- Step: comando individual. Puede ser shell (run:) o una action reutilizable (uses: actions/setup-python@v5).
- Runner: máquina donde corren los jobs. ubuntu-latest (gratis para repos públicos, 2 vCPU/7 GB RAM), windows-latest, macos-latest, o self-hosted (GPU, recursos custom).
- CML (Continuous Machine Learning): CLI de Iterative.ai que postea comentarios en PR con tablas/plots desde el workflow. Hace que las métricas sean visibles al reviewer sin abrir Actions tab.

- OIDC (OpenID Connect): GitHub emite un token efímero firmado que AWS/GCP/Azure aceptan para asumir un rol. Reemplaza guardar AWS_ACCESS_KEY_ID como secret. Una vulnerabilidad menos.
- Required status checks: configurable en Settings → Branches → Branch protection. Lista de jobs cuyo nombre debe terminar en para permitir merge.

Dataset / recursos

- Modelo del ejemplo: RandomForestClassifier sobre Iris (corre en <5 s — ideal para CI).
- Repo template: estructura src/, tests/, .github/workflows/, params.yaml, metrics.json.
- Librerías: scikit-learn, pytest, ruff (lint), cml (npm package).

Ejercicios

1. Workflow mínimo (lint + test): creá .github/workflows/ci.yml con dos jobs en paralelo: lint (corre ruff check) y test (corre pytest). Push y verificá que aparecen los dos checks en el PR.
2. Job de training: agregá un job train que corre python src/train.py, sube model.pkl y metrics.json como actions/upload-artifact. Solo en PRs (if: github.event_name == 'pull_request').
3. Reporte CML: agregá un step que use iterative/setup-cml@v2, escribe report.md con cat metrics.json como tabla, y postea con cml comment create report.md. El comentario aparece en el PR.
4. Comparación contra main: en el mismo job, git fetch origin main && python src/train.py desde main, guardás metrics_main.json, y agregás al reporte una tabla con Δaccuracy, Δf1.
5. Branch protection: en Settings → Branches → Add rule → main, marcá Require status checks to pass before merging y seleccioná lint, test, train. PR con tests rotos = no podés mergear.

Homework verificable

Repo público en GitHub con:

1. Workflow .github/workflows/ml.yml con jobs lint, test, train, report.
2. CML comment funcionando: PR muestra tabla con accuracy, f1, Δaccuracy_vs_main.
3. Cache de pip configurado (actions/setup-python con cache: pip).
4. Un PR abierto donde el modelo degrada accuracy en >3% — el job report debe fallar (exit 1) por umbral configurable en params.yaml.
5. Captura del PR mostrando el comentario CML y los checks rojos.

Criterio de aceptación: el reviewer puede decidir merge/no-merge mirando solo el PR (no Actions tab), y el merge está bloqueado por branch protection cuando algún check falla.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Resource not accessible by integration al	El job no tiene permiso pull-requests: wri
El cache de pip nunca se reutiliza	El cache-key cambia en cada run (probablem
Workflow no se dispara en PRs de forks	Es comportamiento de seguridad: PRs de for
Process completed with exit code 137	OOM (out of memory). El runner gratis tien
OIDC token unavailable al asumir rol AWS	Falta permissions: { id-token: write } en
El job train tarda 25 min en cada push	Falta caché de datasets y modelos pre-entr

Preguntas frecuentes

¿Conviene entrenar el modelo real en CI?

Depende del tamaño. Si el training corre en <10 min en CPU: sí, en GitHub Actions. Si requiere GPU o tarda horas: usá CI solo para correr smoke tests (entrenar 1 epoch en data sample) y dispará el training real en un runner self-hosted con GPU, o en SageMaker/Vertex AI con workflow_dispatch.

¿GitHub Actions vs GitLab CI vs Jenkins?

GH Actions: integrado con GitHub, marketplace enorme de actions, gratis hasta cierto límite para repos públicos. GitLab CI: equivalente para GitLab, con auto-devops más opinado. Jenkins: self-hosted, máxima flexibilidad, mucho mantenimiento. Para equipos en GitHub: Actions sin pensar.

¿Qué hago con secrets de larga vida (API keys, DB passwords)?

(1) Si podés: OIDC + IAM role (zero secrets). (2) Si no: secrets de repo o de organization, rotados cada 90 días vía secrets-manager. (3) Nunca: hardcoded en el YAML ni en el código.

¿Cómo evito que un PR malicioso fugue secrets via CI?

PRs de forks NO ven secrets por default (correcto). Para repos privados con muchos contributors: Settings → Actions → General → Fork pull request workflows from outside collaborators: Require approval for all outside collaborators. Revisar el diff del workflow antes de approve.

¿CML es necesario o uso comentarios manuales con gh pr comment?

CML está diseñado para ML (sabe armar tablas de metrics, embedded plots como PNG base64). gh pr comment es genérico. Para reportes simples: cualquiera. Para comparación de runs con plots: CML.

¿Cómo manejo CI cuando uso self-hosted runners con GPU?

Etiquetá runners (runs-on: [self-hosted, gpu]). Para jobs cortos sin GPU, dejá runs-on: ubuntu-latest. Atención a seguridad: PRs de forks NO deberían correr en self-hosted (pueden ejecutar arbitrary code en tu hardware). Por default GH bloquea esto — no lo deshabilites.

Referencias

- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 10 — Infrastructure and Tooling.
- GitHub Actions docs — empezar por Quickstart y Workflow syntax.
- CML.dev — Continuous Machine Learning de Iterative.ai.
- Configuring OIDC with AWS — deploy sin claves.
- awesome-actions — actions útiles del marketplace.

Siguiente clase

Clase 198 — Docker para empaquetar modelos

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
workflow = '''\
name: ml
on:
  pull_request: { branches: [main] }
  push: { branches: [main] }
permissions:
  contents: read
  pull-requests: write
  id-token: write   # OIDC para AWS
jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: { python-version: "3.12", cache: pip }
      - run: pip install ruff && ruff check src tests
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with: { python-version: "3.12", cache: pip }
      - run: pip install -r requirements.txt && pytest -q
  train-and-report:
    if: github.event_name == 'pull_request'
    runs-on: ubuntu-latest
    needs: [lint, test]
    steps:
      - uses: actions/checkout@v4
# ... (truncado)
```

Archivos complementarios

- notebook.ipynb