

warning "Mise à jour du backend (octobre 2026)" L'inférence GitHub Models a été retirée le 30 juillet 2026. Les références ci-dessous sont conservées uniquement comme contexte historique pendant la migration de ce lab. Utilisez le Lab 013 ? Backends LLM gratuits pour configurer Foundry Local ou Ollama avant de continuer.

Lab 037 : CI/CD pour les agents IA avec GitHub Actions

Niveau : L300 Parcours : Pro Code Durée : ~45 min ? Coût : GitHub Free ? Niveau gratuit
GitHub Actions (2000 min/mois)

Ce que vous apprendrez

Pourquoi les agents IA ont besoin de pipelines CI/CD (ils sont différents des logiciels classiques) Tests de régression de prompts automatisés ? détecter quand un changement de prompt casse le comportement Évaluation LLM-as-judge en CI ? utiliser un modèle pour noter les sorties Patrons de déploiement sécurisé : mode shadow, déploiements canari Un workflow GitHub Actions complet pour un agent IA

Introduction

Livrer des mises à jour d'agents est risqué. Un petit changement de prompt peut silencieusement casser le comportement de l'agent. Contrairement aux logiciels traditionnels, il n'y a pas de compilateur ni de vérificateur de types ? le « bug » est un changement subtil dans la qualité du raisonnement.

Le CI/CD pour les agents IA nécessite :

1. Des tests de régression pour les comportements attendus
2. Des portes de qualité (évaluations automatisées) avant la fusion
3. Des stratégies de déploiement sécurisé qui limitent le rayon d'impact

Prérequis

Compte GitHub (gratuit) GITHUB_TOKEN (Personal Access Token ou secret Actions) Projet Python avec un agent simple (nous en créerons un)

? Fichiers d'accompagnement

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-037/ dans votre répertoire de travail.

Fichier | Description | Télécharger | ai-agent-ci.yml | Modèle de workflow CI/CD | ?

Télécharger |

Exercice pratique

Étape 1 : Créer un module d'agent testable

[diagram or code sample]

agent.py ? un agent de service client simple :

[diagram or code sample]

Étape 2 : Tests comportementaux (rapides, déterministes)

Ces tests vérifient des comportements binaires ? des choses qui doivent toujours être vraies.

tests/test_behaviors.py :

[diagram or code sample]

Étape 3 : Cas de test de référence (JSON)

evals/test_cases.json :

[diagram or code sample]

Étape 4 : Évaluation LLM-as-judge

tests/test_evals.py :

[diagram or code sample]

Étape 5 : Workflow GitHub Actions

.github/workflows/agent-ci.yml :

[diagram or code sample]

Résumé de la stratégie CI

Type de test | S'exécute sur | Vitesse | Coût | Comportemental (pytest) | Chaque PR | Rapide (~10s) | Pas d'appels LLM | Cas de référence | Chaque PR | Moyen (~30s) | Minimal (sortie structurée) | LLM-as-judge | Branche main uniquement | Lent (~2min) | ~10 appels LLM | Déploiement | Main, tests réussis | ? | ? |

tip "Versionnage des prompts" Stockez vos prompts dans des fichiers versionnés (prompts/system.txt), pas en dur dans le code Python. Cela rend les diffs significatifs et permet le rollback.

? Quiz de connaissances

question "Q1 (Exécutez le lab) : Ouvrez lab-037/ai-agent-ci.yml. Combien de jobs sont définis dans le workflow ?"

Ouvrez le fichier et comptez le nombre d'entrées nom-du-job: sous la clé jobs:.

??? success "? Révéler la réponse" 7 jobs

Le workflow définit : unit-tests, integration-tests, security-scan, agent-evaluation, docker-build, deploy-staging et deploy-production. Chaque job a une responsabilité distincte dans le pipeline de déploiement de l'agent.

question "Q2 (Exécutez le lab) : Quel job dans ? ai-agent-ci.yml ne s'exécute PAS sur les pull requests ? uniquement sur les pushes vers la branche main ?"

Regardez la condition if: de chaque job, ou le déclencheur au niveau du workflow vs les conditions par job.

??? success "? Révéler la réponse" integration-tests

Les tests d'intégration appellent de vraies API externes (GitHub Models, etc.) et pourraient échouer en raison de limites de débit ou de coûts s'ils étaient exécutés sur chaque PR. Le job integration-tests a la condition if: github.event_name == 'push' && github.ref == 'refs/heads/main' ? il ne s'exécute que sur les pushes vers main, pas sur chaque pull request. Les tests unitaires s'exécutent sur tous les événements.

question "Q3 (Choix multiple) : Le job deploy-production dans le workflow nécessite une approbation manuelle. Quelle fonctionnalité de GitHub Actions permet cela ?"

- A) Un flag manual-approval: true dans la définition du job - B) Un bloc environment: référençant un environnement protégé avec des réviseurs requis configurés - C) Une étape wait-for-approval utilisant l'API GitHub - D) Définir runs-on: manual au lieu de runs-on: ubuntu-latest

??? success "? Révéler la réponse" Correct : B ? Un environnement GitHub protégé avec des réviseurs requis

Le job deploy-production spécifie environment: production. Dans les paramètres du dépôt GitHub, vous configurez l'environnement production pour exiger 1+ réviseurs spécifiques avant l'exécution du job. Lorsque le workflow atteint ce job, GitHub envoie une notification aux réviseurs, met le workflow en pause et attend. Ce n'est qu'après approbation que le déploiement se poursuit. C'est le patron standard de GitHub Actions pour les déploiements en production avec intervention humaine.

Prochaines étapes

Évaluer à grande échelle avec Azure AI : ? Lab 035 ? Évaluation des agents avec Azure AI

Eval SDK Déployer votre agent sur Azure Container Apps : ? Lab 028 ? Déployer MCP sur Azure Container Apps

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Supporting file: ai-agent-ci.yml

AI Agent CI/CD Pipeline Lab 037: CI/CD for AI Agents (GitHub Actions) This workflow demonstrates a complete CI/CD pipeline for AI agents: 1. Unit tests (mock LLM responses) 2. Integration tests (real GitHub Models API) 3. Agent evaluation (groundedness, coherence, safety) 4. Security scan (prompt injection patterns) 5. Deploy to staging then production Required secrets (Settings ? Secrets ? Actions): GITHUB_TOKEN - auto-provided by GitHub Actions AZURE_CREDENTIALS - Service principal JSON for Azure deployment AZURE_OPENAI_KEY - For production evaluation (L300+)

name: AI Agent CI/CD

on: push: branches: [main] paths: - 'src/' - 'tests/' - 'eval/' pull_request: branches:

[main] paths: - 'src/' - 'tests/'

env: PYTHON_VERSION: "3.11" AGENT_SERVICE_NAME: "outdoorgear-agent" STAGING_URL:

"https://outdoorgear-agent-staging.azurecontainerapps.io"

===== JOBS

===== jobs:

----- # JOB 1: Lint

+ Unit Tests (no API calls, always free) #

----- unit-tests:

name: "? Unit Tests" runs-on: ubuntu-latest

steps: - uses: actions/checkout@v4

- name: Set up Python \${{ env.PYTHON_VERSION }} uses: actions/setup-python@v5 with:

python-version: \${{ env.PYTHON_VERSION }} cache: "pip"

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

pytest-cov pytest-asyncio ruff mypy

- name: Lint with ruff run: ruff check src/ tests/

- name: Type check with mypy run: mypy src/ --ignore-missing-imports

- name: Run unit tests (mocked LLM) run: | pytest tests/unit/ \ -v \ --cov=src \

--cov-report=xml \ --cov-report=term-missing \ -m "not integration"

- name: Upload coverage report uses: codecov/codecov-action@v4 with: file: coverage.xml

fail_ci_if_error: false

----- # JOB 2:

Integration Tests (real API calls ? uses GitHub Models free tier) #

integration-tests: name: "? Integration Tests" runs-on: ubuntu-latest needs: unit-tests #

Only run on main branch pushes, not on every PR to save API quota if: github.event_name ==

'push' && github.ref == 'refs/heads/main'

steps: - uses: actions/checkout@v4

- name: Set up Python \${{ env.PYTHON_VERSION }} uses: actions/setup-python@v5 with:

python-version: \${{ env.PYTHON_VERSION }} cache: "pip"

- name: Install dependencies run: pip install -r requirements.txt pytest pytest-asyncio

- name: Run integration tests env: GITHUB_TOKEN: \${{ secrets.GITHUB_TOKEN }} #

Auto-provided ? no extra secret needed! LLM_BASE_URL:

"https://models.inference.ai.azure.com" LLM_MODEL: "gpt-4o-mini" run: | pytest

tests/integration/ \ -v \ -m "integration" \ --timeout=60

----- # JOB 3:

Security Scan (prompt injection patterns) #

```

----- security-scan:
name: "? Security Scan" runs-on: ubuntu-latest needs: unit-tests
steps: - uses: actions/checkout@v4
- name: Set up Python ${ env.PYTHON_VERSION } uses: actions/setup-python@v5 with:
python-version: ${ env.PYTHON_VERSION } cache: "pip"
- name: Install security tools run: | pip install bandit safety garak
- name: Bandit security scan (Python code) run: bandit -r src/ -f json -o
bandit-report.json || true
- name: Check for known vulnerable dependencies run: safety check --json >
safety-report.json || true
- name: Scan system prompt for injection vulnerabilities run: | python
tests/security/scan_prompts.py \ --prompt-files src/prompts/ \ --output
security-prompt-scan.json
- name: Upload security reports uses: actions/upload-artifact@v4 if: always() with: name:
security-reports path: | bandit-report.json safety-report.json security-prompt-scan.json
# ----- # JOB 4:
Agent Evaluation (requires Azure ? skip if no credentials) #
-----
agent-evaluation: name: "? Agent Evaluation" runs-on: ubuntu-latest needs:
integration-tests if: | github.event_name == 'push' && github.ref == 'refs/heads/main' &&
secrets.AZURE_OPENAI_KEY != "
steps: - uses: actions/checkout@v4
- name: Set up Python ${ env.PYTHON_VERSION } uses: actions/setup-python@v5 with:
python-version: ${ env.PYTHON_VERSION } cache: "pip"
- name: Install evaluation dependencies run: pip install azure-ai-evaluation promptflow
- name: Run agent evaluation suite env: AZURE_OPENAI_KEY: ${ secrets.AZURE_OPENAI_KEY }
AZURE_OPENAI_ENDPOINT: ${ vars.AZURE_OPENAI_ENDPOINT } EVAL_MODEL: "gpt-4o" run: | python
eval/run_evaluation.py \ --dataset eval/eval_dataset.jsonl \ --output-dir eval-results/ \
--metrics groundedness,coherence,relevance,safety \ --fail-threshold 0.8 # Fail if any
metric drops below 80%
- name: Parse evaluation results id: eval_results run: | python eval/parse_results.py
eval-results/summary.json >> $GITHUB_STEP_SUMMARY
- name: Upload evaluation results uses: actions/upload-artifact@v4 if: always() with: name:
eval-results path: eval-results/
- name: Comment evaluation summary on PR if: github.event_name == 'pull_request' uses:
actions/github-script@v7 with: script: | const fs = require('fs'); const summary =
fs.readFileSync('eval-results/summary.json', 'utf8'); const data = JSON.parse(summary);
const body = ## ? Agent Evaluation Results\n\n + | Metric | Score | Threshold | Status |\n
+ |-----|-----|-----|-----|\n + Object.entries(data.metrics).map(([k, v]) =>
| ${k} | ${v.score.toFixed(2)} | 0.80 | ${v.score >= 0.8 ? '?' : '?'} | ).join("\n");
github.rest.issues.createComment({ issue_number: context.issue.number, owner:
context.repo.owner, repo: context.repo.repo, body });
# ----- # JOB 5:
Build Docker image # -----
---- build: name: "? Build Image" runs-on: ubuntu-latest needs: [unit-tests, security-scan]
if: github.event_name == 'push' && github.ref == 'refs/heads/main' outputs: image_tag: ${
steps.meta.outputs.tags } image_digest: ${ steps.build.outputs.digest }
steps: - uses: actions/checkout@v4

```

```

- name: Set up Docker Buildx uses: docker/setup-buildx-action@v3
- name: Log in to GitHub Container Registry uses: docker/login-action@v3 with: registry:
ghcr.io username: ${ github.actor } password: ${ secrets.GITHUB_TOKEN }
- name: Extract metadata id: meta uses: docker/metadata-action@v5 with: images: ghcr.io/${
github.repository }/${ env.AGENT_SERVICE_NAME } tags: | type=sha,prefix=sha-
type=raw,value=latest,enable=${ is_default_branch }
- name: Build and push Docker image id: build uses: docker/build-push-action@v5 with:
context: . push: true tags: ${ steps.meta.outputs.tags } labels: ${
steps.meta.outputs.labels } cache-from: type=gha cache-to: type=gha,mode=max
# ----- # JOB 6:
Deploy to Staging # -----
--- deploy-staging: name: "? Deploy to Staging" runs-on: ubuntu-latest needs: build
environment: name: staging url: ${ env.STAGING_URL }
steps: - name: Azure Login uses: azure/login@v2 with: creds: ${ secrets.AZURE_CREDENTIALS
}
- name: Deploy to Azure Container Apps (Staging) uses:
azure/container-apps-deploy-action@v1 with: appSourcePath: ${ github.workspace } acrName:
outdoorgearacr containerAppName: ${ env.AGENT_SERVICE_NAME }-staging resourceGroup:
rg-outdoorgear-staging imageToDeploy: ghcr.io/${ github.repository }/${
env.AGENT_SERVICE_NAME }:sha-${ github.sha }
- name: Smoke test staging run: | sleep 30 # Wait for deployment curl -f "${
env.STAGING_URL }/health" || exit 1 echo "? Staging deployment healthy"
# ----- # JOB 7:
Deploy to Production (manual approval required) #
-----
deploy-production: name: "? Deploy to Production" runs-on: ubuntu-latest needs:
[deploy-staging, agent-evaluation] if: | always() && needs.deploy-staging.result ==
'success' && (needs.agent-evaluation.result == 'success' || needs.agent-evaluation.result
== 'skipped') environment: name: production url:
"https://outdoorgear-agent.azurecontainerapps.io"
steps: - name: Azure Login uses: azure/login@v2 with: creds: ${ secrets.AZURE_CREDENTIALS
}
- name: Deploy to Azure Container Apps (Production) uses:
azure/container-apps-deploy-action@v1 with: appSourcePath: ${ github.workspace } acrName:
outdoorgearacr containerAppName: ${ env.AGENT_SERVICE_NAME } resourceGroup:
rg-outdoorgear-prod imageToDeploy: ghcr.io/${ github.repository }/${
env.AGENT_SERVICE_NAME }:sha-${ github.sha }
- name: Production health check run: | sleep 30 curl -f
"https://outdoorgear-agent.azurecontainerapps.io/health" || exit 1 echo "? Production
deployment successful"
- name: Create GitHub deployment status uses: actions/github-script@v7 with: script: |
github.rest.repos.createDeploymentStatus({ owner: context.repo.owner, repo:
context.repo.repo, deployment_id: context.payload.deployment?.id, state: 'success',
description: 'Deployed to production', environment_url:
'https://outdoorgear-agent.azurecontainerapps.io' });

```