DEVOPS & DÉPLOIEMENT

CI/CD avec GitHub Actions

Automatiser les tests, la qualité du code et les déploiements directement depuis GitHub. Workflows YAML, runners, secrets, environnements et déploiement continu vers un VPS ou une plateforme cloud : le prolongement naturel de la formation Git & GitHub pour passer du commit au déploiement sans intervention manuelle.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
3j (21h)	~~900.00 €~~ **540.00 €**	**Intermédiaire**	**Français**	**1–6**	**Formation**

1 PUBLIC VISÉ

- Développeurs web (Node.js, PHP, Python) maîtrisant Git et GitHub et souhaitant automatiser leurs tests et leurs déploiements.
- Développeurs ayant suivi la formation Git et GitHub : gestion de versions pour débutants ou disposant d'une expérience équivalente avec les branches, les pull requests et les dépôts distants.
- Équipes de développement souhaitant standardiser leur workflow de livraison et supprimer les déploiements manuels.
- Freelances cherchant à livrer des projets plus fiables grâce à des pipelines d'intégration et de déploiement continus.
- Développeurs ayant suivi la formation Docker pour développeurs web et souhaitant automatiser le build et le push de leurs images.

2 PRÉREQUIS

- Maîtrise de Git et GitHub : commits, branches, merges, pull requests, dépôts distants. La formation Git et GitHub : gestion de versions pour débutants est recommandée ou une expérience équivalente.
- Avoir un projet web fonctionnel dans au moins un langage (Node.js, PHP ou Python) avec une suite de tests basique existante ou en cours de création.
- Notions de base sur le terminal Linux et les variables d'environnement.
- Notions de base sur le format YAML : indentation, listes, dictionnaires.
- La formation Docker pour développeurs web est un plus mais n'est pas obligatoire — le Module 5 sur Docker dans les pipelines est accessible sans expérience Docker préalable.

3 OBJECTIFS PÉDAGOGIQUES

- Comprendre les concepts CI/CD : intégration continue, livraison continue, déploiement continu
- Maîtriser la syntaxe YAML des workflows GitHub Actions : events, jobs, steps, runners
- Automatiser les tests unitaires et d'intégration sur chaque push ou pull request
- Gérer les secrets et les variables d'environnement de façon sécurisée dans les workflows
- Construire et publier des images Docker depuis un pipeline GitHub Actions
- Déployer automatiquement sur un VPS via SSH ou sur une plateforme cloud (Render, Railway)
- Utiliser les environnements GitHub pour gérer les déploiements staging et production
- Optimiser les workflows : cache des dépendances, jobs parallèles, conditions et matrices

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) - Fondamentaux GitHub Actions et premiers workflows

Module 1 - Comprendre la CI/CD et l'écosystème GitHub Actions (2h)

CI/CD : concepts et valeur concrète (45min)

- L'intégration continue : détecter les régressions au plus tôt sur chaque modification du code
- La livraison continue : maintenir le code toujours déployable, automatiser jusqu'au staging
- Le déploiement continu : aller jusqu'en production sans intervention humaine
- Ce que la CI/CD change concrètement : fin des conflits de merge douloureux, des déploiements manuels à risque
- L'écosystème CI/CD : GitHub Actions, GitLab CI, CircleCI, Jenkins

Architecture de GitHub Actions (1h15)

- Les composants clés : workflow, event, job, step, action, runner
- Les runners : GitHub-hosted (ubuntu, windows, macos) vs self-hosted
- Le répertoire .github/workflows/ : convention de nommage, un fichier par workflow
- La syntaxe YAML : indentation stricte, types de valeurs, chaînes multilignes
- L'interface GitHub Actions : onglet Actions, visualisation du pipeline, logs en temps réel
- Les minutes gratuites : quotas par plan, runners payants, optimiser la consommation

Cas pratique : créer le premier workflow hello.yaml, déclenché sur push, explorer l'interface GitHub

Module 2 - Syntaxe YAML et déclencheurs (2h)

Les événements déclencheurs (1h)

- push : déclencher sur des branches spécifiques, exclure des chemins avec paths-ignore
- pull_request : déclencher sur l'ouverture, la mise à jour, la synchronisation d'une PR
- workflow_dispatch : déclencher manuellement depuis l'interface avec des inputs
- schedule : déclenchement planifié avec la syntaxe cron
- release : déclencher sur la publication d'une release GitHub
- Combiner plusieurs événements avec la liste on

Jobs, steps et contextes (1h)

- Déclarer un job : runs-on, steps, needs pour les dépendances entre jobs
- Les actions officielles : actions/checkout, actions/setup-node, actions/setup-python, actions/setup-php
- Les commandes shell dans les steps : run single-line vs multiline avec |
- Les contextes GitHub Actions : github, env, steps, runner
- Passer des données entre steps : outputs, variables d'environnement dans $GITHUB_ENV
- Les conditions : if sur un job ou un step, success(), failure(), github.ref

Cas pratique : pipeline à deux jobs séquentiels, job "lint" puis job "build" déclenché uniquement si lint passe

Module 3 - Automatiser les tests (3h)

Pipeline de tests pour une application Node.js (1h)

- Configurer l'environnement Node.js : actions/setup-node avec la version cible
- Installer les dépendances avec npm ci pour la reproductibilité
- Lancer les tests : npm test avec Jest ou Vitest
- Linter le code : ESLint dans le pipeline
- Vérifier le formatage : Prettier en mode --check
- Publier les résultats de tests : rapport JUnit avec actions/upload-artifact

Cas pratique : pipeline complet sur une API Express, checkout, setup Node 20, npm ci, eslint, jest

Pipeline de tests pour PHP et Python (1h)

- PHP : configurer shivammathur/setup-php, installer avec Composer, lancer PHPUnit, analyser avec PHPStan
- Python : configurer actions/setup-python, installer avec pip, lancer pytest, linter avec flake8 ou ruff
- Gérer les dépendances de test : base de données de test, variables d'environnement
- Services GitHub Actions : lancer un conteneur PostgreSQL ou MySQL comme service annexe

Cas pratique : pipeline Python avec pytest et une base PostgreSQL comme service

Optimiser le pipeline de tests (1h)

- Cache des dépendances : actions/cache pour node_modules, pip, Composer
- La stratégie matrix : tester sur plusieurs versions de Node.js ou PHP en parallèle
- Jobs parallèles : lint, tests unitaires et tests d'intégration en simultané
- Fail-fast : arrêter tous les jobs de la matrice dès qu'un échoue
- Artifacts : uploader les rapports de couverture, les captures d'écran de tests E2E

Cas pratique : refactorer le pipeline Node.js, ajouter le cache npm, une matrice Node 18/20/22, mesurer le gain de temps

Jour 2 (7h) - Secrets, environnements et déploiement continu

Module 4 - Gérer les secrets et les variables d'environnement (1h30)

Secrets GitHub et variables de configuration (45min)

- Pourquoi ne jamais stocker de secrets dans le code ou dans les logs de workflow
- Les secrets GitHub : repository secrets, organisation secrets, environment secrets
- Déclarer et utiliser un secret dans un workflow : ${{ secrets.MON_SECRET }}
- Les variables GitHub Actions (vars) : pour les valeurs non sensibles, réutilisables entre workflows

Les secrets masqués dans les logs : comportement par défaut, précautions supplémentaires

Cas pratique : workflow qui utilise une clé API stockée en secret pour appeler un service externe

Environnements GitHub : staging et production (45min)

- Les environnements GitHub : créer "staging" et "production" dans les paramètres du dépôt
- Les protection rules : approbation manuelle requise avant un déploiement en production
- Les secrets par environnement : des credentials différents pour staging et production
- Référencer un environnement dans un job : environment: production
- L'historique de déploiement : visualiser qui a déployé quoi, quand, avec quel statut

Cas pratique : configurer deux environnements, staging automatique et production avec approbation requise

Module 5 - Construire et publier des images Docker (2h)

Build et push d'image Docker depuis GitHub Actions (1h)

- L'action docker/build-push-action : la référence pour les pipelines Docker
- S'authentifier sur un registre depuis le workflow : Docker Hub, GitHub Container Registry
- Construire et tagger l'image avec le SHA du commit et le tag latest
- Le cache de layers Docker dans GitHub Actions : cache-from et cache-to
- Le multi-platform build : produire des images linux/amd64 et linux/arm64

Cas pratique : pipeline qui construit une image Docker, la pousse sur ghcr.io avec tag SHA et tag latest

Scanner les images et les dépendances (1h)

- Pourquoi scanner : CVE dans les images de base, dépendances avec des failles connues
- Scanner les dépendances npm/pip/composer : npm audit, pip-audit, Dependabot
- Scanner l'image Docker avec Trivy : intégration dans le pipeline, seuil de sévérité, rapport SARIF
- Publier le rapport de sécurité dans l'onglet "Security" de GitHub
- Dependabot : automatiser les mises à jour de dépendances

Cas pratique : ajouter une étape Trivy, rapport publié dans GitHub Security, pipeline bloqué si CVE critique

Module 6 - Déploiement continu sur un VPS (2h)

Déployer sur un VPS via SSH (1h)

- Générer une paire de clés SSH dédiée au déploiement
- Stocker la clé privée dans les secrets GitHub, installer la clé publique sur le VPS
- Se connecter au VPS depuis le workflow avec appleboy/ssh-action
- Commandes de déploiement : docker compose pull + docker compose up -d
- Vérifier le déploiement : healthcheck HTTP après le redémarrage des conteneurs

Rollback manuel : revenir à l'image précédente en cas d'échec du healthcheck

Cas pratique : pipeline complet, push sur main, tests, build image, push ghcr.io, déploiement SSH, healthcheck

Déployer sur une plateforme cloud - Render, Railway (1h)

- Render : déploiement depuis une image Docker via l'API Render
- Railway : déploiement depuis le dépôt GitHub ou depuis une image
- Les variables d'environnement sur ces plateformes : synchroniser avec les secrets GitHub Actions
- Stratégie blue-green simplifiée sur Render : zero-downtime deployment
- Comparer les approches : VPS vs plateforme cloud

Cas pratique : déploiement automatique sur Render déclenché depuis le pipeline GitHub Actions

Module 7 - Workflows avancés et réutilisabilité (1h30)

Composite actions et reusable workflows (1h30)

- Les actions composites : factoriser une séquence de steps récurrente dans une action réutilisable locale
- Créer une action composite dans .github/actions/ : action.yaml, inputs, outputs
- Les reusable workflows : partager un workflow entier entre plusieurs dépôts avec workflow_call
- Passer des inputs et des secrets à un workflow réutilisable
- Les actions du Marketplace : évaluer une action tierce, étoiles, mainteneur, code source, permissions
- Épingler les actions à un commit SHA pour prévenir les supply chain attacks

Cas pratique : extraire la séquence setup + cache + install dans une action composite locale

Jour 3 (7h) - Qualité du code, bonnes pratiques et projet de synthèse

Module 8 - Qualité du code et protection des branches (2h)

Branch protection rules et statut checks (1h)

- Les branch protection rules sur main : bloquer le push direct, exiger des PR
- Les required status checks : empêcher le merge d'une PR si le workflow CI est rouge
- Les required reviews : combiner validation humaine et validation automatique
- La règle "dismiss stale reviews" : invalider les approbations après un nouveau commit
- CODEOWNERS : assigner automatiquement des reviewers selon les fichiers modifiés

Cas pratique : configurer les branch protection rules, tenter de pousser directement sur main, vérifier que le merge est bloqué

Couverture de code et reporting qualité (1h)

- Générer un rapport de couverture : Istanbul/c8 pour Node.js, coverage.py pour Python, Xdebug pour PHP
- Publier la couverture dans les commentaires de PR avec actions/github-script

Intégrer Codecov ou Coveralls : badge de couverture dans le README, historique des tendances

- CodeQL : analyse statique de sécurité gratuite pour les dépôts publics
- Afficher les annotations de lint directement dans les diffs de PR

Cas pratique : pipeline qui génère un rapport de couverture Jest, le publie en commentaire sur la PR, bloque si couverture sous 70%

Module 9 - Débogage, monitoring et bonnes pratiques (1h)

Déboguer et optimiser ses workflows (1h)

- Activer le debug logging : ACTIONS_RUNNER_DEBUG et ACTIONS_STEP_DEBUG
- Re-run un job en mode debug depuis l'interface GitHub
- Tester localement avec act : simuler GitHub Actions sur sa machine avant de pousser
- Mesurer la durée des jobs : identifier les étapes lentes, optimiser en priorité
- Les concurrency groups : annuler un run en cours quand un nouveau est déclenché
- Recevoir des notifications sur l'état des workflows : email, Slack via webhook

Cas pratique : activer le debug sur un workflow existant, identifier l'étape la plus lente, ajouter un concurrency group

Module 10 - Atelier préparatoire au projet (1h)

Conception du pipeline de synthèse

- Choisir le projet : apporter une application existante ou utiliser un projet fourni
- Cartographier le pipeline cible : événements, jobs, étapes, environnements
- Identifier les secrets nécessaires et leur portée
- Choisir la cible de déploiement : VPS via SSH ou plateforme cloud
- Estimer la durée du pipeline et identifier les optimisations à prioriser

Module 11 - Projet de synthèse (3h)

Pipeline CI/CD complet de bout en bout

Livrables attendus : au moins deux workflows distincts (CI sur PR, CD sur merge dans main), cache des dépendances actif, secrets correctement gérés, déploiement automatique fonctionnel sur staging, approbation requise pour la production

- Étape 1 : workflow CI, tests, lint, couverture, rapport sur la PR
- Étape 2 : workflow CD, build image Docker, push registre, déploiement staging automatique
- Étape 3 : protection de la branche main et gate de production avec approbation
- Étape 4 : optimisation, cache, jobs parallèles, concurrency group

Présentation du pipeline (10min) : démonstration d'un push qui déclenche CI puis CD, revue collective des choix

Retour formateur individualisé sur la structure des workflows, la gestion des secrets et la stratégie de déploiement

5 COMPÉTENCES VISÉES

- Écrire des workflows GitHub Actions structurés et maintenables en YAML
- Automatiser les tests, le lint et la couverture de code sur chaque pull request
- Gérer les secrets et les variables d'environnement de façon sécurisée par environnement
- Construire et publier des images Docker depuis un pipeline avec cache de layers
- Déployer automatiquement sur un VPS ou une plateforme cloud avec gate de production
- Optimiser un pipeline : cache des dépendances, jobs parallèles, actions composites réutilisables

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel. Le formateur alterne entre méthode démonstrative (construction live de workflows sur un dépôt GitHub réel avec des applications Node.js, PHP et Python), méthode interrogative (analyse des logs d'échec et diagnostic collectif) et méthode active (exercices d'écriture de pipelines et projet de synthèse fil rouge). L'accent est mis sur la pratique immédiate : chaque workflow est déclenché, observé et corrigé dans la même session.

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

- Support de cours numérique mis à disposition
- Fichiers de workflows YAML annotés avec exercices et corrections par module
- Environnement : compte GitHub + VS Code + accès à un VPS Ubuntu pour les déploiements
- Pour le distanciel : visioconférence, partage d'écran, chat en direct
- Accès à la plateforme pédagogique LaPolaris (supports, ressources, émargement)

8 MODALITÉS D'ÉVALUATION

- En cours de formation : pipelines construits et testés en direct à chaque module
- En fin de formation : pipeline CI/CD complet sur un projet réel avec déploiement automatique
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- Workflows GitHub Actions fonctionnels avec déclencheurs, jobs et steps correctement structurés
- Cache des dépendances actif et jobs parallèles réduisant le temps de pipeline
- Secrets et variables d'environnement correctement isolés par environnement GitHub
- Pipeline CI bloquant le merge d'une PR en cas d'échec des tests ou du lint
- Déploiement automatique fonctionnel sur staging avec gate de production opérationnel
- Actions composites ou reusable workflows réduisant la duplication entre pipelines

10 MODALITÉS DE VALIDATION

Attestation de fin de formation délivrée à l'issue du parcours, conditionnée à une assiduité d'au moins 80 % et à la réalisation du projet de synthèse. L'attestation précise les objectifs atteints et les compétences acquises.

11 SUIVI ET ACCOMPAGNEMENT

- Feuilles d'émargement signées par demi-journée (présentiel) ou émargement numérique (distanciel)
- Traçabilité des activités pédagogiques réalisées
- Attestation d'assiduité délivrée en fin de formation
- Suivi individuel via la vérification du dépôt GitHub de chaque apprenant en fin de parcours

12 CONDITIONS D'ACCÈS

Formation accessible sur inscription directe, sans prérequis administratif particulier. Le financement peut être pris en charge par l'employeur dans le cadre d'un plan de développement des compétences, ou en autofinancement. LaPolaris est un organisme de formation en cours de certification Qualiopi.

13 DÉLAIS D'ACCÈS

Inscription possible jusqu'à **5 jours ouvrés** avant le début de la session. Pour toute demande urgente, nous contacter directement.

ACCESSIBILITÉ · HANDICAP

Nos formations sont accessibles aux personnes en situation de handicap. Pour toute situation nécessitant un aménagement (matériel, temporel ou pédagogique), nous vous invitons à nous contacter avant l'inscription afin d'étudier les adaptations possibles. Référent handicap : contact@lapolaris.fr

LaPolaris

TÉL. +33762584798

EMAIL contact@lapolaris.fr

WEB lapolaris.fr