DevOps

Intégration Continue

Thomas Ropars

thomas.ropars@univ-grenoble-alpes.fr

2021

Agenda

DevOps

Définition (simple)

Ensemble de techniques et d'outils facilitant le passage du développement à la production.

Bien plus que ça:

- Modèle de fonctionnement de l'entreprise
  - Impliquant tous les maillons de la chaîne (RHs, finances, etc.)
- Modèle d'interactions entre les équipes
- Intégration du retour sur expérience
- Une "culture"

DevOps

Relation entre Dev et Ops:

- Dev: Équipes de développeurs logiciels
- Ops: Équipes en charge de la mise en production des produits

Antagonisme fort:

- Dev: Modifications aux moindres coûts, le plus rapidement possible
- Ops: Stabilité du système, qualité

L'automatisation est au cœur de l'approche DevOps

DevOps: Automatisation

Intégration continue

Une méthode de développement logiciel dans laquelle le logiciel est reconstruit et testé à chaque modification apportée par un programmeur.

Livraison continue

La livraison continue est une approche dans laquelle l'intégration continue associée à des techniques de déploiement automatiques assurent une mise en production rapide et fiable du logiciel.

Déploiement continu

Le déploiement continu est une approche dans laquelle chaque modification apportée par un programmeur passe automatiquement toute la chaîne allant des tests à la mise en production. Il n'y a plus d'intervention humaine.

Agenda

Introduction

Intégration continue

Livraison continue

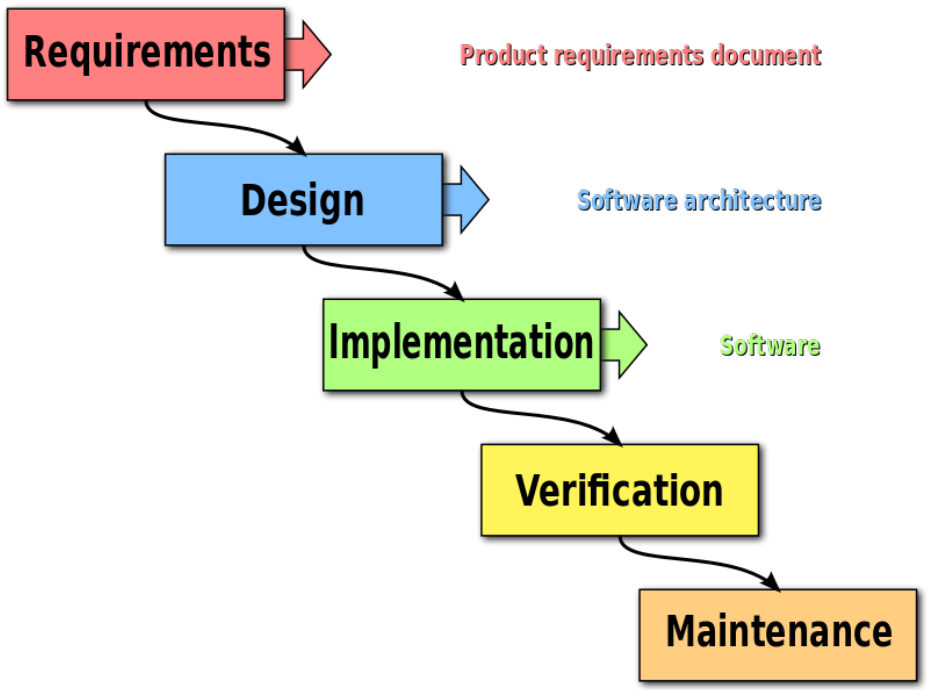
Pipeline de CI/CD

Contexte

Figure de P. Kemp / P. Smith

Cycle de développement logiciel

- Modèle de base en cascade
  - Exigences bien définies initialement
- Peut être bien adapté pour des logiciels critiques (avion, centrale nucléaire, . . .)



Agile

Inadapté aux entreprises agiles (startups):

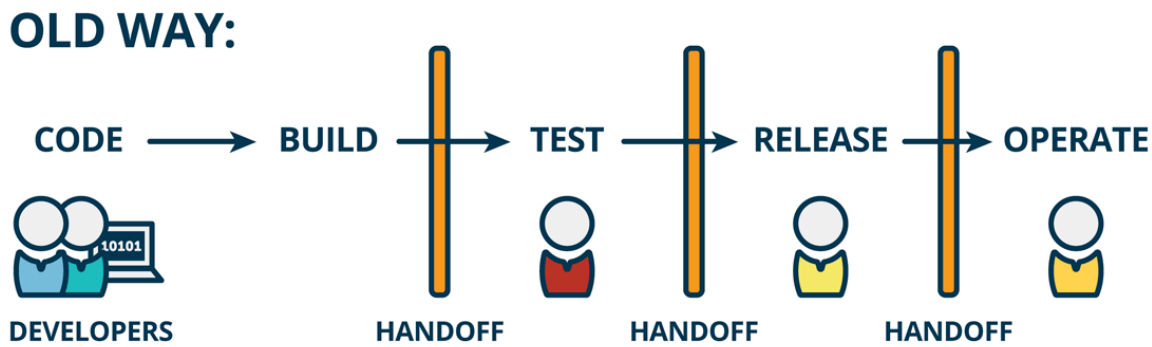
- Contexte turbulent
- Exigences changeantes
- Clients inconnus (attente du marché inconnue)

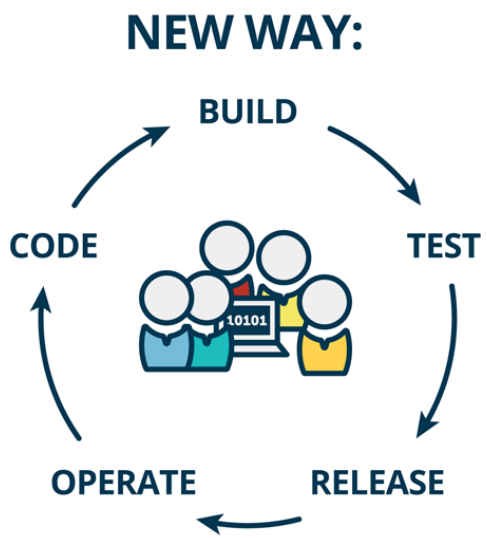
Développement agile

- Extreme Programming
- Relation étroite entre le client et l'équipe de dev.
  - Développement de scénario
  - Test Driven Development
  - Changement de priorité, ...

Évolution en schéma

Figure from `https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/`





L'intégration continue

L'intégration continue fait référence à plusieurs pratiques:

- Construire une version fonctionnelle du système chaque jour
- Exécuter les tests tous les jours
- *Committer* ses changements sur le dépôt tous les jours
- Un système qui observe les changements sur le dépôt et si il détecte un changement:
  - Récupère une copie du logiciel depuis le dépôt
  - Compile et exécute les tests
  - Si les tests passent, possibilité de créer une nouvelle *release* du logiciel
  - Sinon averti le développeur concerné

Pourquoi c'est utile: un exemple

Votre chef vous avertit:

- Dans une heure il passe au bureau avec des investisseurs pour voir la version la plus récente de votre système.

2 scénarios possibles

- Un grand moment de stress qui finit en un échec
  - L'état du prototype n'est pas clair, le merge est difficile, la démo ne marche pas.
- Un non évènement qui se passe sans accroc
  - L'état du prototype est clair, la démo se passe sans soucis

A propos de l'intégration continue

Avoir un logiciel prêt au déploiement

- Au début de la phase de développement, le déploiement peut sembler lointain
  - Ceci peut être source de problèmes
- L'intégration continue oblige à avoir un système qui fonctionne
  - Peut-être qu'il ne fait rien
  - Ajout des fonctionnalités de manière incrémentale

Échouer au plus tôt

- Compiler/Tester le système plusieurs fois par jour doit être la règle
- Détecter les erreurs au plus tôt permet de réagir vite

A propos de l'intégration continue

Interactions entre équipes de développement

- Toutes les équipes travaillent sur le même dépôt
- L'ensemble des composants logiciels peuvent être testés ensemble
  - Les problèmes d'intégration sont mis en évidence au plus tôt

Détecter les régressions

- Une fonctionnalité cesse de fonctionner correctement après une modification
- Relancer l'ensemble des tests à chaque mise à jour permet de détecter efficacement ces problèmes

Un exemple

La *disparition* de la société de trading *Knight Capital*

Les faits

- Société leader dans le *high-frequency* trading
- Par erreur, 4 millions de transactions exécutées en quelques minutes le 1er Août 2012
- Plus de 400M$ de perte

Les causes

- Une erreur dans la mise à jour des serveurs de l'entreprise
- Échec du code supposé contrôler la validité des transactions
  - Code *cassé* lors d'une mise à jour précédente
  - Pas de tests de régression

Mise en place

Pour faire de l'intégration continue, nous avons besoin de:

Mise en place

Pour faire de l'intégration continue, nous avons besoin de:

- Un dépôt pour le code source
- Un processus de construction automatique du logiciel
- Une plateforme pour exécuter des tests

Nous avons besoin aussi de:

- Une volonté de travailler de manière incrémentale
- Une procédure commune pour envoyer les modifications

Procédure d'envoi des modifications

Tous les développeurs doivent suivre la procédure suivante:

1. Démarrer de la version la plus récente du système
2. Écrire des procédures de test et faire les changements voulus
3. Exécuter tous les tests et s'assurer qu'ils passent
4. Récupérer les dernières modifications depuis le dépôt, relancer les tests, et s'assurer qu'ils passent
5. Envoyer les contributions vers le dépôt

À ce moment, les tests d'intégration continue vont être exécutés automatiquement.

Les outils

Les outils que nous pouvons utiliser:

- Un dépôt pour le code source
  - svn, git, ...
  - Github
- Un processus de construction automatique du logiciel
  - Make, Ant, Maven ...
- Une plateforme pour exécuter des tests
  - xUnit, JUnit ...
  - Jenkins, Gitlab CI/CD, Github Actions ...
  - Docker

Agenda

Livraison continue

Continuous Delivery

Définition

- Optimisation des procédures de livraison de logiciels
  - Objectif: Un logiciel doit pouvoir être mis en production à tout moment
  - Les procédures de livraison doivent être automatisées

Motivation

- La mise en production d'un logiciel est une phase critique et complexe
- Le faire le plus souvent possible permet de réduire les risques et de mieux maîtriser cette procédure
- Nécessite un maximum d'automatisation

Livraison continue et Lean startup

Lean startup

- *Démarrer maigre*

Livraison continue et Lean startup

Lean startup

- *Démarrer maigre*
- Évaluer l'intérêt d'une fonctionnalité en investissant le moins d'effort possible.
- Peut s'appliquer aux startups mais aussi aux entreprises *classiques*

La livraison continue plutôt que les Releases

- Dans le modèle *classique*, la mise en production d'une nouvelle fonctionnalité se fait lors d'une release
  - Les releases ne sont en général pas très fréquentes
- La livraison continue permet de mettre de nouvelles fonctionnalités rapidement en production

Un exemple de Lean startup

source: A practical guide to continuous delivery

- Contexte: Un site d'achat en ligne
- Nouvelle fonctionnalité: Choisir le jour de livraison d'une commande

Les étapes

- Faire de la pub pour la nouvelle fonctionnalité (le développement n'a pas commencé)
- Si beaucoup de clicks, mettre en oeuvre la fonctionnalité (service basique)
  - La livraison continue offre un avantage compétitif
- Superviser l'utilisation de la nouvelle fonctionnalité
  - Imaginer des améliorations pour la fonctionnalité et les mettre en production rapidement

Les étapes de la livraison continue

Les étapes principales d'une procédure de livraison continue sont:

1. Commit d'une modification
2. Exécution des tests unitaires (Intégration continue)
3. Tests de validation fonctionnelle (acceptance test)
   - Tests black-box
   - Testent si le logiciel répond bien au besoin du client
   - Peuvent aussi être automatisés
4. Tests de performances
   - Testent si le logiciel peut répondre à la charge
   - Performance et passage à l'échelle
5. Tests exploratoires
   - Tests non-automatisés effectués par des experts
   - Exemple: tests d'utilisabilité
6. Le déploiement en production

La gestion de l'infrastructure

Dans l'approche "livraison continue", la gestion de l'infrastructure est elle aussi automatisée.

Gestion automatisée

- La mise en place de l'infrastructure d'exécution est du code
  - Géré par le gestionnaire de versions
  - Décrit les différentes étapes de la mise en place (par ex: les logiciels à installer)
- Des environnements d'exécutions doivent être mis place à différentes étapes de la livraison continue
  - Pour toutes les phases de test
  - Pour la mise en production

Gestion automatisée de l'infrastructure

Infrastructure as code

Avantages

Gestion automatisée de l'infrastructure

Infrastructure as code

Avantages

- Limite les risques d'erreurs
- Assure d'avoir toujours le même environnement d'exécution
- Permet de tracer les modifications et de reproduire les erreurs
- Permet de lier les modifications de l'infrastructure et les modifications de l'application
- Permet de déployer rapidement un environnement d'exécution

Les outils

- Docker et les Dockerfile
- Les outils permettant la configuration d'un environnement logiciel (à l'aide de recettes):
  - Chef, Puppet, Ansible, etc.
- Les outils permettant d'approvisionner en ressources (machines virtuelles ou conteneurs)
  - Vagrant, Terraform, etc.

Minimisation des risques

La mise en production est une opération risquée.

Minimisation des risques avec le livraison continue

- Chaque fonctionnalité est mise en production de manière indépendante
  - Des *petites* mises à jour limitent les risques de problèmes
- L'approche *Infrastructure as code* limite les mauvaises surprises
  - Tests dans les conditions de l'environnement de production
  - Procédures automatisées

La mise en production

Quelques remarques

- Phase critique
- Revenir en arrière n'est pas toujours facile
  - L'utilisation de gestionnaires de version facilite ce point
  - Problèmes de retrocompatibilité
- Important de faire des *smokes tests* avant une utilisation réelle en production
  - Vérifier que l'application est démarrée et *fonctionne*

Les techniques de mise en production

Canary release

- Rendre disponible une nouvelle version du logiciel à un petit sous-ensemble des utilisateurs (10%):
  - Tester en conditions réelles
  - Avoir un retour sur la nouvelle version (intérêt de nouvelles fonctionnalités)
  - Plusieurs nouvelles versions d'une fonctionnalité peuvent être testées en parallèle.
  - On peut vouloir sélectionner certains clients spécifiques pour cette tester une nouvelle version

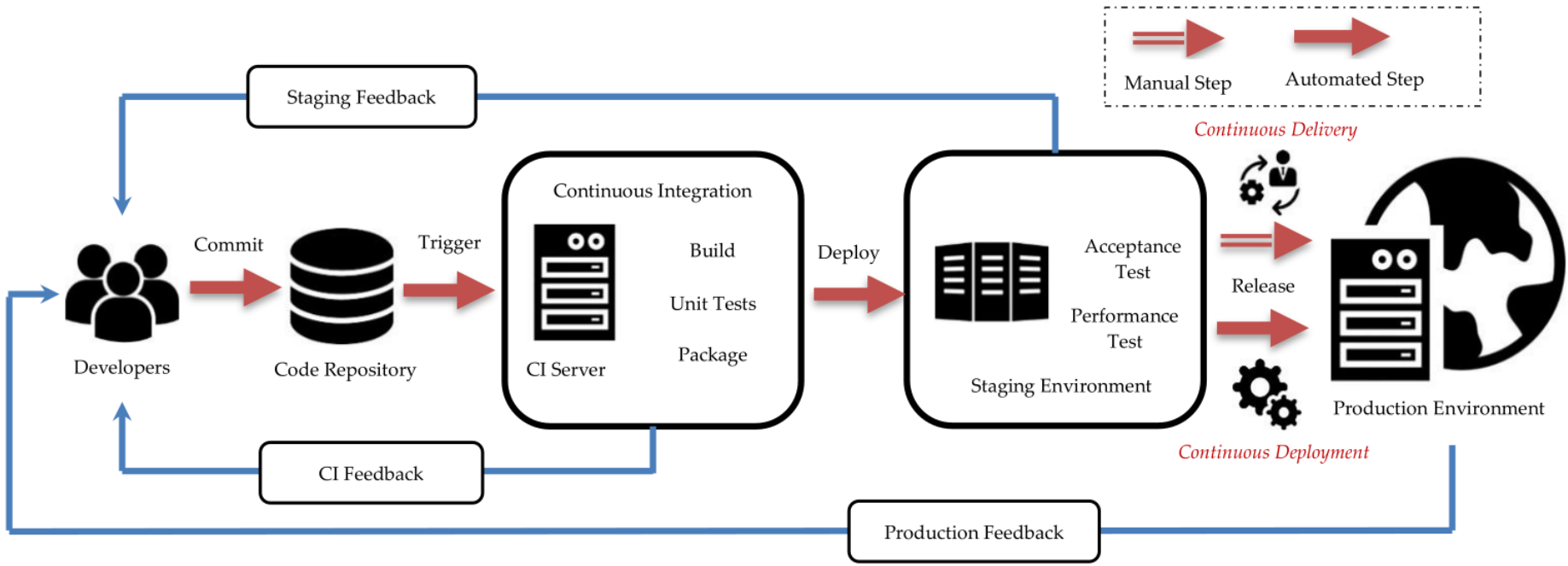
Les techniques de mise en production

Blue-Green deployment

- Avoir deux environnements de production actifs en même temps (Bleu et Vert), un seul est utilisé.
- Déploiement et derniers tests du nouveau logiciel sur l'environnement non utilisé
- Changement de version active en changeant le routage des requêtes.
  - Changement de version très rapide
  - Retour en arrière facile

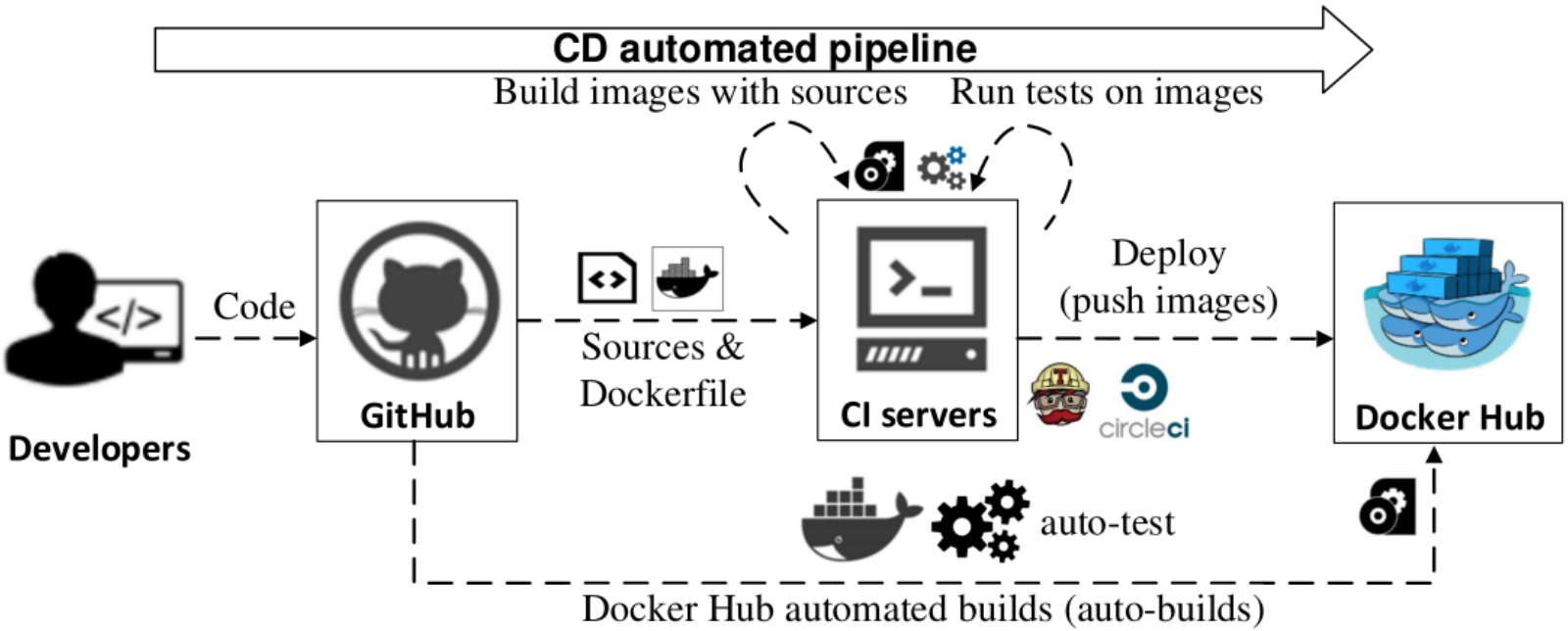
Résumé intégration/livraison/déploiement continu

Figure by Shahin et al.



Workflow de CD et Docker

Figure by Zhan et al.



2 options

- L'outil de CI/CD publie une nouvelle image quand les tests passent avec succès
- Dockerhub surveille votre dépôt pour construire une nouvelle image lorsqu'il y a des modifications.

Agenda

Support CI/CD avec Github ou Gitlab

Fonctionnement similaire offert par Github et Gitlab

- Définition d'un pipeline d'intégration/livraison continu
- Définition à partir d'un fichier Yaml
  - Gitlab: Fichier `gitlab-ci.yml` à la racine du dépôt
  - Github: Fichier yaml à stocker dans le répertoire `.github/workflows/`

Points d'entrée pour la documentation

- Gitlab: `https://docs.gitlab.com/ee/ci/quick_start/`
- Github: `https://docs.github.com/en/actions/learn-github-actions/introduction-to-github-actions`

Les concepts principaux

- Un workflow (pipeline) est une procédure automatisée associée à votre dépôt.
- Un workflow est composé de un ou plusieurs jobs
  - Les jobs peuvent s'exécuter en parallèle par défaut
  - Possibilité de définir des dépendances entre les jobs (Concept de stage)
- Un job est composé de steps
  - Une étape est une action/commande
- Un job est exécuté par un Runner
  - Un Runner est un serveur
    - Gitlab/Github fournissent des Runners
    - On peut aussi héberger ses Runners
  - Les jobs sont exécutés sur de Runners différents
- Des events déclenchent le lancement d'un workflow

Exemple de workflow Gitlab

```
stages:
  - build
  - test
  - deploy

build_a:
  stage: build
  script:
    - echo "This job will run first."

build_b:
  stage: build
  script:
    - echo "This job will run first, in parallel with build_a."

test_ab:
  stage: test
  script:
    - echo "This job will run after build_a and build_b have finished."

deploy_ab:
  stage: deploy
  script:
    - echo "This job will run after test_ab is complete"
```

Exemple de workflow Gitlab

Commentaires sur le workflow précédent

- Les jobs sont associés à des *stages* pour définir des dépendances
- Les job sont: `build_a`, `build_b`, `test_ab`, etc.
  - Les jobs sont exécutés dans des contextes d'exécution indépendants
- Les *steps* sont définies sous la balise `script`

Même exemple avec Github

https://docs.github.com/en/actions/learn-github-actions/migrating-from-gitlab-cicd-to-github-actions

```yaml
jobs:
  build_a:
    runs-on: ubuntu-latest
    steps:
      - run: echo "This job will be run first."

  build_b:
    runs-on: ubuntu-latest
    steps:
      - run: echo "This job will be run first, in parallel with build_a"

  test_ab:
    runs-on: ubuntu-latest
    needs: [build_a,build_b]
    steps:
      - run: echo "This job will run after build_a and build_b have finished"

  deploy_ab:
    runs-on: ubuntu-latest
    needs: [test_ab]
    steps:
      - run: echo "This job will run after test_ab is complete"
```

Spécifier une image Docker à utiliser pour un job

GitLab CI/CD

```
my_job:
  image: node:10.16-jessie
```

GitHub Actions

```
jobs:
  my_job:
    container: node:10.16-jessie
```

Références

- Notes de D. Donsez
- Notes de K. M. Anderson

Continuous Delivery (pour aller plus loin)

- Eric Ries, The Lean Startup: `http://theleanstartup.com/`
- Dzone Guide To Continuous Delivery: `https://dzone.com/guides/continuous-delivery-3`
- Eberhard Wolff, A practical guide to continuous delivery