Automatisation de la chaîne de production

F. Lassabe

IUT Nord-Franche-Comté

Qu'attendez-vous du cours ?

Lien direct

Introduction

- Créer un logiciel/service logiciel
- De nombreuses étapes
  - Définition du besoin
  - Analyse économique
  - Définition technique
    - Totalement spécifiée (cycle V, etc.)
    - Direction globale spécifiée incomplètement (agile)
  - Implémentation
  - Validation
    - Tests unitaires
    - Tests d'intégration
  - Déploiement
  - Facturation
  - Fin de vie
- Tout ceci forme la gestion du cycle de vie du logiciel (PLM, Product Lifecycle Management)

Complexité

- Nombre d'étapes
- Répétition de certaines (dev, tests unitaires, tests d'intégration, éventuellement déploiement)
- Erreurs/oublis
  - Ennui
  - Fatigue
  - Interruptions
- Chaque étape nécessite des manipulations ...
- sauf avec automatisation

Un monde sans automatisation

- Imaginez
- Le développeur écrit son code
- Puis il lance manuellement ses tests unitaires (s'il y pense)
- Puis il fait ses tests d'intégration (s'il y pense)
- Puis il déploie (s'il y pense)
- Les oublis sont presque garantis
- Une méthode : organisation
  - Le code écrit est copié sur un support amovible
  - Le développeur va à une borne dédiée
  - Insère le support de stockage
  - Puis procède aux étapes test unitaires, d'intégration, déploiement
- Fastidieux mais a plus de chances de succès

Une solution audacieuse

- Une fois le code écrit, il est livré
- Uniquement si vous aimez vous fâcher avec les clients

La vraie solution

- Automatiser la chaîne de production
- Automatiser :
  - Les actions répétitives et identiques
  - Gérées par des programmes
  - Lancés aux modifications du code
- Pourquoi automatiser ?
  - Pas de la magie (automatiser du code pourri reste du code pourri)
  - Intégration de notion de qualité
  - Capacité à vérifier la conformité fonctionnelle du code
  - Comparable au contrôle qualité + automates des chaînes industrielles

CI/CD

- Deux significations
  - Continuous Integration/Continuous Delivery (plus commun)
    - Quand le code est poussé
    - Il est compilé
    - Et testé
    - Puis (delivery) : proposé en test au client et déployé si validé
  - Continuous Integration/Continuous Deployment (plus complexe)
    - Comme la première définition, mais déploiement automatisé de bout en bout
    - Plus complexe : empêcher les bugs d'atteindre la production

Exemples de logiciels

- Gitlab
  - On travaillera avec Gitlab
  - Logiciel reconnu, couplé avec gestion du code
- Jenkins
- Travis CI
- Et bien d'autres

Les désavantages

- Coût de mise en place/maintien (compétence + RH)
- Coût de structuration autour des pipelines
  - Suites de tests pertinentes et complètes
- Applications critiques
  - Conformité (sûreté/sécurité)
  - Relecture pour certification
- Pas d'outil *silver bullet*, tout est question de compromis
- Environnements hybrides possibles
  - Ce qui peut être automatisé l'est
  - Le reste ne l'est pas

Avant tout un outil

- Comme tout outil
  - La performance dépend de l'utilisation
  - L'utilité dépend du cadre
- Performance
  - Performance
    - Bien configuré
    - Bien utilisé (ce pour quoi il est prévu)
- Utilité
  - **Tout** doit partir de la politique d'organisation !
  - Un outil n'est bon que s'il s'inscrit dans cette politique.
- Généralement : la chaîne CI/CD s'intègre bien dans le développement logiciel

CI/CD dans Gitlab

- Plusieurs notions/outils
- Gitlab
  - Forge logicielle (libre)
  - Implémente git avec une une surcouche web
- Les runners Gitlab
  - Des machines communicant avec gitlab
  - Fournissent des ressources pour la CI/CD
- Les pipelines
  - Définis dans les projets avec CI/CD
  - Composés de jobs
  - Exécutés dans des conteneurs
- Les jobs
  - Une action d'un pipeline
  - Par ex.: une suite de tests, un build, etc.
- Les stages
  - Regroupent des jobs dans la séquence

Ordre d'exécution

- Dans un pipeline
- Les jobs d'un stage
  - sont exécutés en parallèle
- Les stages d'un pipeline
  - Exécutés les uns après les autres
- Ordre initial
  - build
  - test
  - deploy

En pratique

- Pipeline nommé `.gitlab-ci.yml` à la racine du projet
- Syntaxe YAML (cf. extension)
- Identifie plusieurs étapes (stages)
  - Étapes et leurs ordres
- Si pas défini, stages vaut par défaut :

```
stage:
  - .pre
  - build
  - test
  - deploy
  - .post
```

- Un pipeline avec seulement .pre et .post ne s'exécutera jamais
- Il faut **au moins une** étape "réelle"

Jobs

- Définis comme des sections du fichiers YAML
- Nommés différemment des mots réservés (stages, variables, workflow, etc.)
- Exemple :

```
pdflatex:
  stage: build
  script:
    - ln -s texmf /workdir/texmf
    - export TEXMFHOME=$PWD/texmf
    - make
  artifacts:
    paths:
      - 0_UFC/R5.09/intro-virtualisation.pdf
      - 0_UFC/R5.09/virtualbox-networking.pdf
      - 0_UFC/R5.09/web-servers.pdf
      - 0_UFC/R5.09/intro-proxmox.pdf
```

Les jobs

- Les jobs sont l'unité de base du pipeline
- Leur rôle est de produire quelque chose
- Ils possèdent un certain nombre de paramètres
- Notamment ce qui leur permet d'agir :
  - image : l'image docker utilisée pour le job
  - artifacts : les fichiers créés et conservés
  - script : les scripts à exécuter par le job
  - stage
  - variables : les variables nécessaires au job

Passons à la pratique

- Trouver une instance gitlab avec CI/CD
- Suivre le sujet de TP disponible sur Moodle