Intégration continue

Principes, Apport et Mise en place

Sommaire

- Introduction
- Qu’est-ce que l’intégration continue
- Pourquoi l’utiliser
- Comment la mettre en place
- Conclusion

Introduction

- Déploiement d’un projet sans CI/CD
  - Équipe dédiée au déploiement
  - Tests effectués avant le déploiement par l’équipe
  - Éventuel backup de la BD
  - Téléversement du produit sur le serveur via divers moyens
    - Ansible, Docker, ssh, RDP, FTP, etc.
  - Downtime prolongé le temps de la mise à jour (Si non-redondance)
  - Risque d’erreur lors du déploiement (Résultant donc en un downtime)

- Déploiement continue / Livraison continue ?

Qu’est-ce que l’intégration continue

- But de l’intégration continue
  - Automatiser
  - Permettre aux développeurs de déployer d’eux même (DevOps)
  - Ne pas oublier des étapes lors du déploiement
  - Réduction du downtime

- Coût ?
  - Faible
    - Paiement à la minute pour les services SaaS
    - Possibilité de l’héberger sur ses propres serveurs
    - Potentiellement une équipe de moins nécessaire

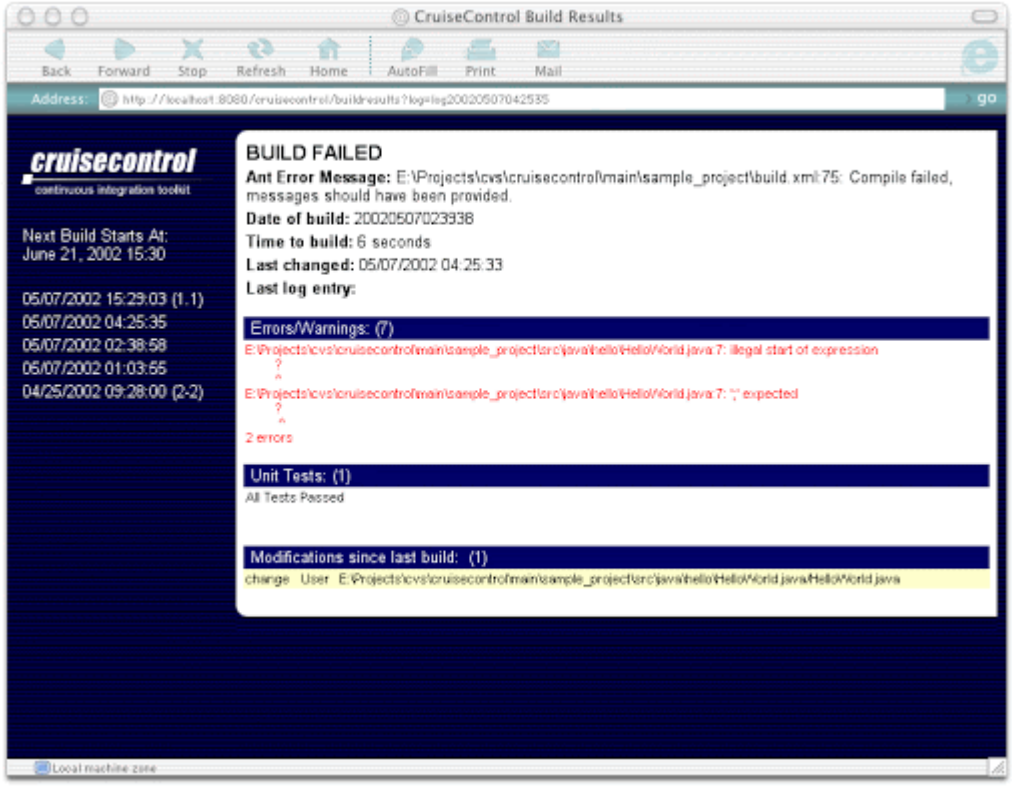
Qu’est-ce que l’intégration continue

- Outils historiques
  - CruiseControl : Premier outil d’intégration continue
- Outils populaires
  - Jenkins
  - Gitlab CI/CD
  - Github Actions
  - CircleCI

Qu’est-ce que l’intégration continue

- CruiseControl
  - Créé en 2001, en Java
  - Système de notifications par courriel
  - Intégration facile avec Ant
  - Tableau de bord en JSP pour avoir des informations sur l’état des build

- Des outils inspirés ont vu le jour par la suite :
  - CruiseControl.NET (.NET)
  - CruiseControl.rb (Ruby)



Qu’est-ce que l’intégration continue

Jenkins

- Jenkins
  - Créé comme fork de Hudson (2005)
  - Première version stable : 2011
  - Remplacement de CruiseControl
  - Le plus populaire

Qu’est-ce que l’intégration continue

- Gitlab
  - Frontend web pour Git intégrant des fonctionnalités supplémentaire
  - Concurrent à Github, Bitbucket, ...
  - Propose un système d’intégration continue intégré

Pourquoi l’utiliser ?

- Automatisation
  - Suppression de taches répétitives
  - Aucun risques d’oubli
  - Donne le pouvoir au développeurs
  - Gain de temps
  - Économies
    - Mise en production « sûre » (S’arrête si les tests ne passent pas)
    - Pas forcément d’équipe / personne dédiée
  - MEP plus fréquentes

Pourquoi l’utiliser ?

- Inconvénients à prendre en compte
  - Changement des processus habituels
  - Serveurs supplémentaires (Si non-SaaS)
  - Nécessite des processus adapté (Doit automatiser le build avant de pouvoir l’utiliser en CI/CD)
  - Si plusieurs développeurs ajoutent du code en même temps, il ne seront pas forcément dans la même build

Mise en place

- Prenons un code simple en Golang

main.go

```go
package main

import "fmt"

func main() {
    fmt.Println(Addition(5, 2))
}

func Addition(a, b int) int {
    return a+b+1
}
```

main_test.go

```go
package main

import "testing"

func TestAddition(t *testing.T) {
    resultat := Addition(8, 2)
    if resultat ≠ 10 {
        t.Errorf("Addition(8, 2) = %v, want 10", resultat)
    }
}
```

```
---  ~/Git/exemple_cicd <master*(0) M> » go test
--- FAIL: TestAddition (0.00s)
    main_test.go:8: Addition(8, 2) = 11, want 10
FAIL
exit status 1
FAIL    github.com/oxodao/exemple_cicd  0.002s
```

Mise en place

- Utilisons Github Actions
  - Nous créons tout d’abord un fichier .github/workflows/build.yml

```yaml
on:
  create:
    tags:
      - v*

jobs:
  build:
    runs-on: ubuntu-latest
    if: github.actor == github.event.repository.owner.login
    steps:
      - name: Install Go
        uses: actions/setup-go@v2
        with:
          go-version: 1.15.2

      - uses: actions/checkout@master

      - name: tests
        run: go test
```

Se déclenche quand un tag est poussé sur Git

On installe le compilateur

On récupère le code

On lance les tests

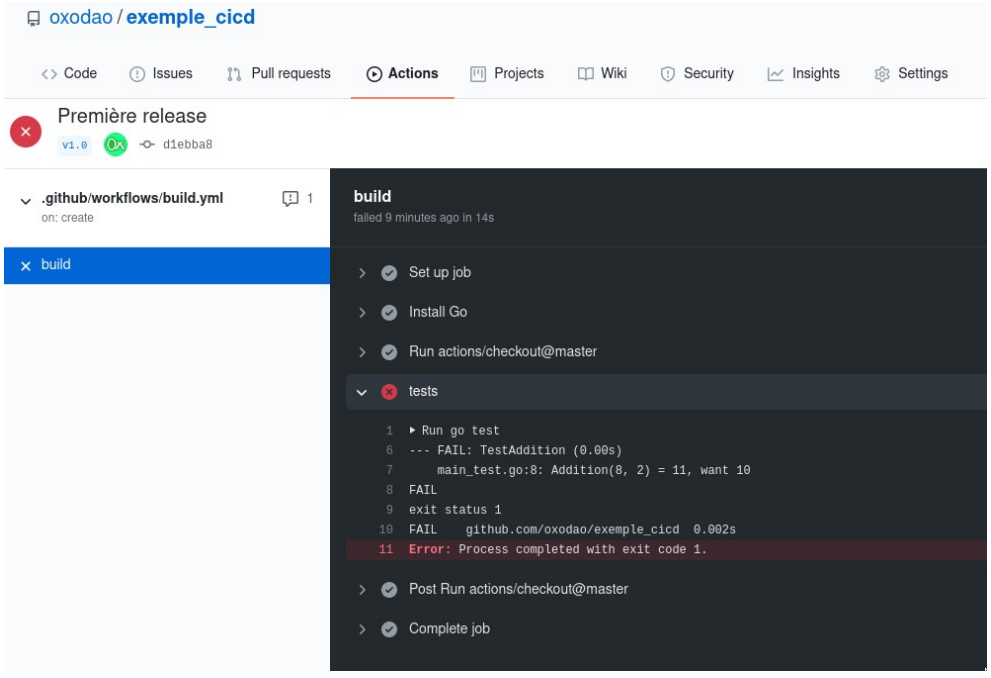
Mise en place

- Exécution des tests
  - Dans notre fichier nous avons spécifié que les tests étaient exécutés a chaque fois que l’on créait un nouveau tag

```
---  ~/Git/exemple_cicd ‹master*(0) M› » git add .
---  ~/Git/exemple_cicd ‹master*(0) M› » git commit -m "Première release"
[master d1ebba8] Première release
 1 file changed, 1 insertion(+), 1 deletion(-)
---  ~/Git/exemple_cicd ‹master(1)› » git push -u origin master
---  ~/Git/exemple_cicd ‹master(0)› » git tag -a "v1.0" -m "Première release"
---  ~/Git/exemple_cicd ‹master(0)› » git push --follow-tags
```

Mise en place

- Affichage des résultats
  - L’onglet « Action » de notre dépôt affiche les résultats du CI/CD



Mise en place

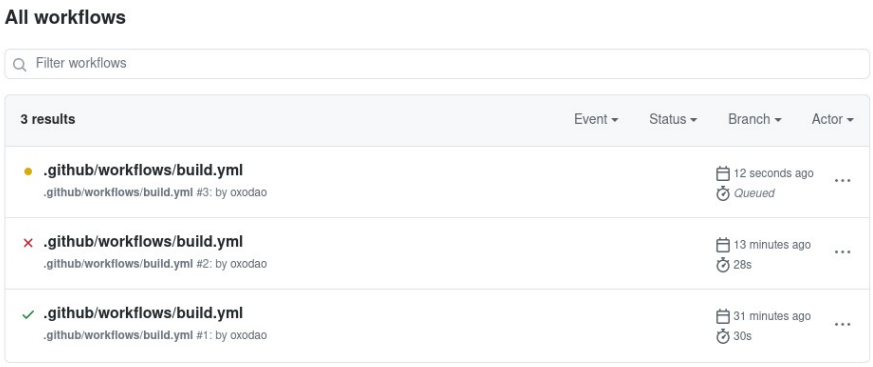
- Corrigeons notre code et créons un nouveau tag

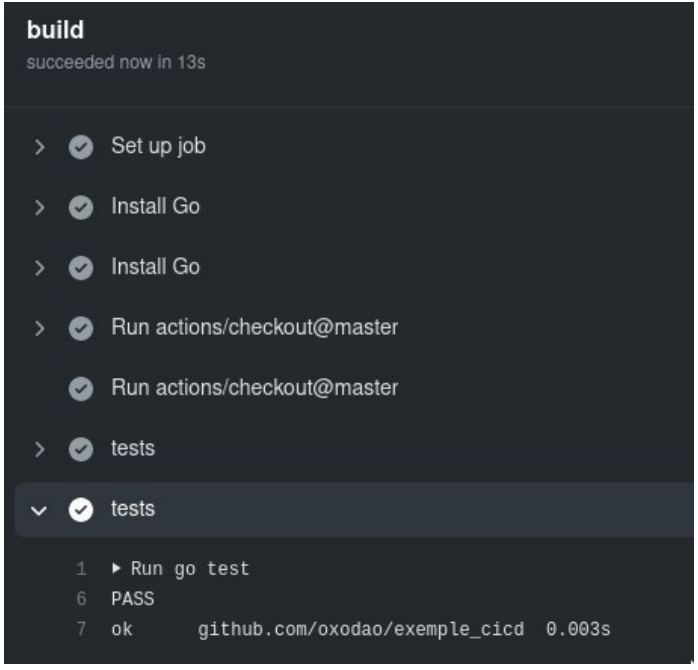
```go
package main

import "fmt"

func main() {
    fmt.Println(Addition(5, 2))
}

func Addition(a, b int) int {
    return a+b+1
}
```





Conclusion

- Intégration continue : que prendre en compte lors de son choix de plateforme ?
  - Hébergement actuel de notre VCS
  - Technologie utilisée
  - Coût
  - Hébergée ? SaaS ?

Conclusion

- Liens
  - Projet démo
    - https://github.com/oxodao/exemple_cicd
  - Présentation
    - https://janczewski.fr/cicd.pdf

Merci de votre attention

Des questions ?