Intégration Continue

Description pas à pas d’un déploiement local

Introduction **1**

1. Installer Jenkins **2**

2. Créer un “build job” **4**

3. Lancer “build” après un “git commit” **5**

4. Installer fastlane **6**

5. Intégrer fastlane **9**

6. Automatiser les tests **10**

7. Créer un ‘Github Issue’ **11**

8. Conclusion **12**

Bibliographie **13**

Introduction

Ce bonus vise à mettre en place un outil d'intégration continue. L'objectif est d'automatiser les tests.

Nous utiliserons git pour déclencher des actions dans jenkins.

Nous allons installer et configurer localement jenkins et y intégrer fastlane.

Environnement, septembre 2018 :

- mac OS 10.13.6
- jenkins 2.141
- fastlane 2.104.0

1. Installer Jenkins

1.1 Prérequis

Avec brew :

- installer java 8

```
brew tap caskroom/versions
brew cask install java8
```

- Installer docker (optionnel)

```
brew cask install docker
```

1.2 Installer jenkins

```
brew install jenkins
```

1.3 Démarrer jenkins

Entrer la ligne de commande `jenkins` dans une nouvelle fenêtre du terminal

> Ne pas utiliser `brew services start jenkins` parce que cela crée un problème avec `$PATH` : jenkins ne reconnait pas les commandes lancées depuis des scripts bash (les 'build' échouent avec `command not found`)

Si besoin on pourra redémarrer jenkins depuis le navigateur en entrant dans la barre des adresses : `localhost:8080/safeRestart` (pour une installation en local ; mettre l'adresse adéquate lorsque jenkins est installé sur un serveur distant).

Pour quitter jenkins : `^+c` dans la fenêtre du terminal où l'on a lancé jenkins (stoppe le serveur)

1.4 Accéder à jenkins

Lancer le navigateur à l'adresse `localhost:8080`

Installer les plug-ins par défaut et créer un utilisateur

_

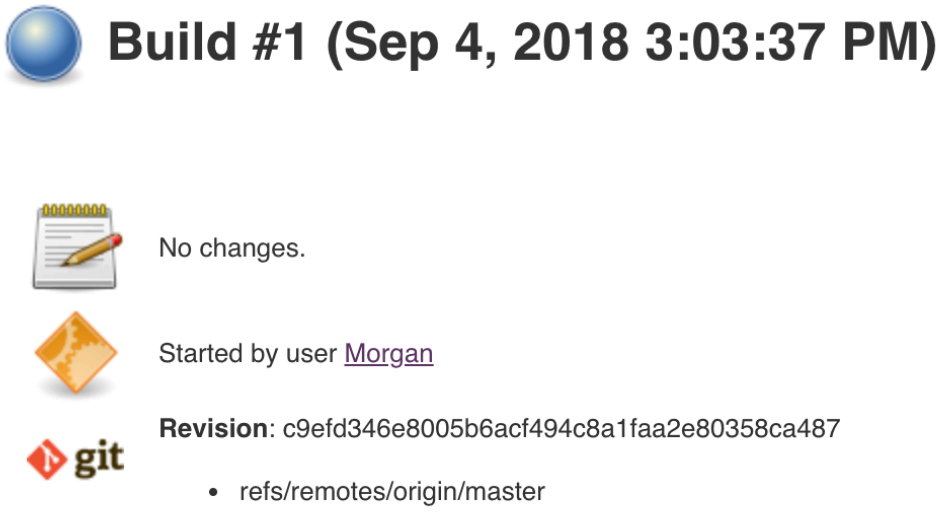
Pour résoudre les problèmes de logIn, s'il y a lieu :

- stopper jenkins
- ouvrir `~/.jenkins/config.xml`
- changer `<useSecurity>true</useSecurity>` pour `<useSecurity>false</useSecurity>`
- start jenkins
- ouvrir le navigateur à localhost:8080 (ou l'adresse courante de jenkins)
- dans `jenkins/manage jenkins/configure global security` rétablir la sécurité
- entrer à nouveau le mot de passe dans la page de l'utilisateur
- log in (le nom de l'utilisateur doit apparaître dans la barre de navigation de jenkins)
- log out puis log in pour vérifier

2. Créer un “build job”

(en local)

1. Créer un *freestyle project*
2. cocher *git* dans *sourcecode management* et ajouter l’adresse locale du dépôt
3. *save*
4. *build*
5. si le *build* a réussi, dans *statut* un point bleu s’affiche



note : le projet se trouve copié dans le workspace jenkins.

3. Lancer “build” après un “git commit”

1. Permettre à jenkins d’écouter les déclencheurs (triggers)

Cocher *Poll SCM* dans `job/<nom du projet>/configure/build triggers` (laisser ’schedule’ vide)

2. Pour créer un déclencheur, créer un fichier exécutable nommé ‘post-commit’

```
cd ~/path/to/mon/projet/.git/hooks
touch post-commit
```

3. Créer le script

Dans le nouveau fichier :

```
#!bin/sh

curl http://your_jenkins_server/git/notifyCommit?url=<URL of the Git repository>
```

Par exemple, avec jenkins en local : `http://127.0.0.1:8080/git/etc`.

L’URL du dépôt est le chemin vers le projet lorsque l’on travaille localement (sinon renseigner l’URL du dépôt distant).

Pour le rendre executable par jenkins si besoin, modifier les droits d’exécution :

```
chmod +x ~/path/to/mon/projet/.git/hooks/post-commit
```

4. Installer fastlane

1. Prérequis :

Installer bundler : `sudo gem install bundler`

2. Installer fastlane

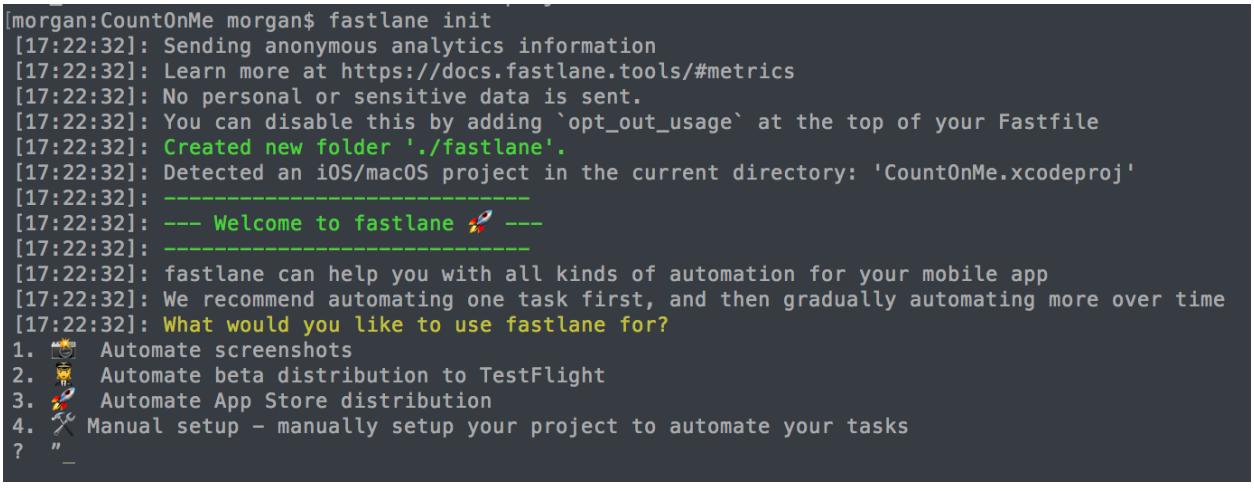
`brew cask install fastlane --no-quarantine`

(utiliser le flag `--no-quarantine`: cf fastlane issue @ github)

Ajouter `export PATH="$HOME/.fastlane/bin:$PATH"` au bash profile : ~/.bashprofile

3. Initialiser fastlane dans le répertoire du projet

Entrer la commande `fastlane init`



4. Définir des variables d'environnement

Ajouter `export LC_ALL=en_US.UTF-8` et `export LANG=en_US.UTF-8` dans `~/.bash_profile`

5. Tester fastlane

- Ouvrir `mon projet/fastlane/Fastfile`
- Y ajouter :

```
fastlane_version "2.24.0"
default_platform :ios

platform :ios do
      lane :example do
            puts("this is my first 🚀 lane")
            end
      end
```

- Entrer la commande `fastlane example`

6. Utiliser Gemfile

(Cela permet de définir la version et les dépendances de fastlane, et d'accélérer l'utilisation de fastlane.)

- Ouvrir ./Gemfile (ou créer ce fichier) et ajouter : `source "https://rubygems.org"` `gem "fastlane"`
- Entrer la commande `sudo bundle update`
- Ajouter Gemfile et Gemfile.lock à l'index de git : `git add Gemfile Gemfile.lock`
- Utiliser par la suite la commande `bundle exec fastlane [lane]`

7. Tester avec `bundle exec fastlane example`

```
morgan:CountOnMe morgan$ bundle exec fastlane example
[08:21:41]: ----------------------------------------
[08:21:41]: --- Step: Verifying fastlane version ---
[08:21:41]: ----------------------------------------
[08:21:41]: Your fastlane version 2.103.1 matches the minimum requirement of 2.24.0 ✅
[08:21:41]: ------------------------------
[08:21:41]: --- Step: default_platform ---
[08:21:41]: ------------------------------
[08:21:41]: Driving the lane 'ios example' 🚀
[[08:21:41]: this is my first 🚀 lane

+------+---------------------+-------------+
|             fastlane summary             |
+------+---------------------+-------------+
| Step | Action              | Time (in s) |
+------+---------------------+-------------+
| 1    | Verifying fastlane  | 0           |
|      | version             |             |
| 2    | default_platform    | 0           |
+------+---------------------+-------------+

[08:21:41]: fastlane.tools finished successfully 🎉
```

5. Intégrer fastlane

1. Créer un script bash pour chacune des commandes utilisées précédemment :

```
#!bin/bash
bundle install
```

```
#!bin/bash
bundle exec fastlane example
```

2. Modifier les permissions des scripts (si besoin) :

```
chmod a+x monScript.sh
```

3. Ajouter un “build step : Execute Shell” dans l’onglet “build environment” du projet jenkins

4. Entrer la commande d’exécution

```
bash -ex ./path/vers/monscript.sh
```

5. Reprendre les étapes 3 et 4 pour chaque script que l’on veut voir exécuter

Dans notre exemple, nous créons deux *build step* : le premier pour *bundle install* le deuxième pour *fastlane*

6. Sauver et tester en cliquant ‘Build Now’

La sortie de console de jenkins doit être comparable à la sortie du tty.

6. Automatiser les tests

1. Ajouter à la Fastfile :

```
lane :tests do
  run_tests(scheme: "MyAppTests")
end
```

(Cela remplace la lane *example*)

2. Créer un script fastlaneTests.sh :

```
#!bin/bash
bundle exec fastlane tests
```

3. Dans jenkins/ExecuteShell, entrer la commande :

```
bash -ex ./path/vers/fastlaneTests.sh
```

4. Configurer Products/MonAppliTests.xctests scheme :

- sélectionner la target MonAppliTests
- sélectionner Product -> scheme -> edit sheme
- sélectionner Run
- choisir MonAppli.app comme Executable
- cocher Shared (ou sélectionner Manage scheme, ajouter MonAppliTests et cliquer sur Shared)

5. Dans jenkins, 'Build Now'

7. Créer un ‘Github Issue’

Pour créer automatiquement un ‘Github Issue’ lorsque le build échoue :

1. Dans le projet jenkins, onglet General, cocher ‘Github project’ et renseigner l’URL du dépôt distant

2. Dans l’onglet Post-build Actions, Add post-build action

Sélectionner *Create Github issue on failure*

3. Tester en faisant échouer le build. Le Github Issue est automatiquement clos dès que le build passe à nouveau.

8. Conclusion

Nous pouvons donc lancer automatiquement les tests de notre application après chaque commit sur la branche locale git 'master'.

Lorsque le 'build' de jenkins échoue, une issue est ouverte sur le dépôt distant du code de l'application. L'issue créée met à disposition la sortie de console de jenkins sur le dépôt distant. L'issue est automatiquement close lors du prochain 'build' réussi.

Ainsi nous avons appris à mettre en place localement un serveur jenkins et à y intégrer le service fastlane, en lien avec le système de gestion de version git. Nous avons tous les éléments pour déployer ces services d'intégration continue au flux de travail d'une équipe de développeurs et pour adapter l'automatisation de tâches aux besoins de l'équipe. Dans ce cadre il serait pertinent de déployer jenkins sur un serveur distant.

Nous avons également mis en place de solides fondations pour intégrer la distribution continue aux processus de travail.

Bibliographie

Jenkins

- Jenkins
- Installing Jenkins

Fastlane

- fastlane - App automation done right
- Setup - fastlane docs

Bundler

- Bundler: The best way to manage a Ruby application's gems
- GitHub - bundler/bundler: Manage your Ruby application's gem dependencies

Git hooks

Automatically triggering a Jenkins build on Git commit ~ andyfrench.info

Git - Git Hooks Jenkins_GitPlugIn

Intégration Fastlane et Jenkins

- Continuous Integration and Delivery for iOS with Jenkins and Fastlane (Part 1)
- iOS Continous integration: Xcode Server, Jenkins, Travis and fastlane
- Installing Jenkins on OS X Yosemite — Nick Charlton

Shell script et Jenkins

- build*and*test*examples/Shell.md at master · softwaresaved/build*and*test*examples · GitHub
- Run a bash shell script - macOS - SS64.com
- Command not found issue

Docker

- Docker CE