

TP

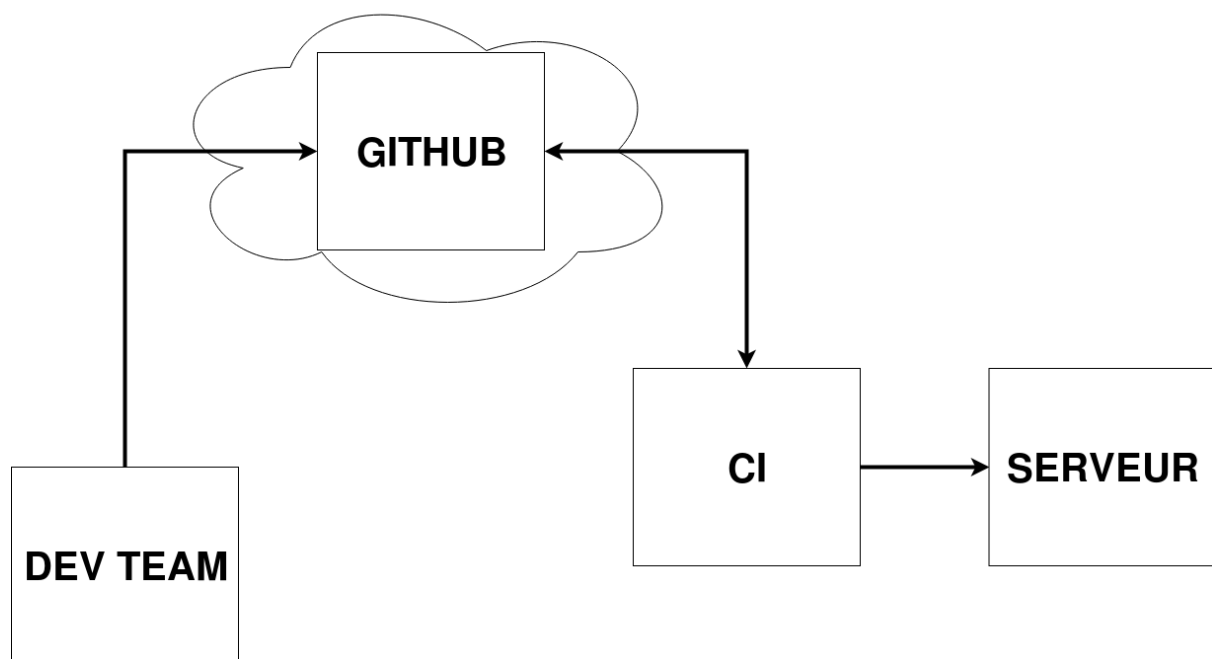
Objectif:

Le but de ce TP est de construire un pipeline delivery. Ce pipeline doit tester, reconstruire en continue le code à chaque modification et doit déployer l'application.

Il faut avoir 2 VM:

- La CI + scripts deploiement
- La machine de l'application

Le TP tourne sur python 2.7, aucune compatibilité n'est garanti sur python3.X



La dev team est une équipe composé de développeur qui va envoyer son code sur Github.

Lorsque le code est sur Github, on veut que le serveur de CI (Continuous Integration) lance:

1. Télécharge le repository sur Github. Soit votre CI est exposé sur internet, github peut envoyer un event via un hook. Si ce n'est pas le cas il faut que la CI puisse se connecter au repository pour inspecter si il y a des modifications sur le repository toutes les X minutes. Le workflow de CI doit se déclencher automatiquement lors qu'il y a une modification sur Github.
2. Lorsque la CI a réussi a télécharger le repository les tests peuvent être lancés
3. Si les tests échouent la CI arrête son workflow et envoie une notification sur la pull request pour indiquer le fail

4. Si les test sont en succès, il faut envoyer une notification sur la pull request pour indiquer l'état du build et déclencher le job de build
5. Le job de build va construire un package wheel et le stocker dans pypicloud
6. Si le build se passe bien envoyer une notification sur la pull request et déclencher le job de déploiement

Composition des serveurs:

- CI:
  - Un client git
  - Un serveur Jenkins
  - Un serveur Pypicloud (en entreprise on va le mettre sur une machine dédiée)
  - Scripts de déploiement (normalement il ne sont pas géré par la CI mais dans le cadre du TP on va laisser jenkins piloter le déploiement)
- SERVEUR:
  - C'est la machine qui va faire tourner l'application qui a été packagé dans le pypicloud
  - Elle a besoin d'un fichier de configuration pour télécharger les artefacts depuis pypicloud
  - Il y a un serveur nginx pour exposer la bonne version du service

Les étapes du TP sur le sujet du pipeline de delivery:

- Mise en place d'un repository github
- Installation d'un serveur CI (jenkins):  
<https://jenkins.io/doc/book/installing/#debian-ubuntu>
- Installer pip: curl --silent --show-error --retry 5  
<https://bootstrap.pypa.io/get-pip.py> | sudo python
- Mettre en place le repository d'artefact (pypicloud), lors de la phase de setup il vous proposera soit filesystem soit S3, choisissez filesystem  
<https://github.com/stevearc/pypicloud>
- Connecter le repository git vers la CI pour trigger les builds
- Pour créer les jobs regardez qu'est ce qu'un jenkinsfile  
<https://jenkins.io/doc/book/pipeline/jenkinsfile/>
- Mettre en place le job de test
  - Créer un job sur jenkins appelé Test
  - Ce job se déclenche sur les push de commit sur toutes les branches pas que master et sur les merge de pull request
  - Il faut que les dépendances python s'installe à chaque fois sans que l'utilisateur jenkins soit sudoer dans un virtualenv (Il faudra régler les problèmes de dépendances, les packages python doivent être installés avec pip, le package manager de python, il y a quelques dépendances

systèmes à installer avec apt. Pas besoin de mettre dans le job de CI pour l'installation des dépendances système

- Lancer les tests avec la commande `py.test [dossier]`, ex `py.test tests` (vous avez quelques chose à faire avec le `PYTHONPATH` pour que les tests puissent s'exécuter)
- 
- [Optionnel] mettre en place des notifications sur la pull request sur Github (api de notification) (cf annexe)
- Mettre en place le job de build, il faut construire un package de type wheel
  - Fichier de configuration pour l'upload:  
[http://pypicloud.readthedocs.io/en/latest/topics/getting\\_started.html#uploading-packages](http://pypicloud.readthedocs.io/en/latest/topics/getting_started.html#uploading-packages)
  - Recherche sur internet comment construire un package de type wheel, on ne veut pas des packages de type egg
  - Pour l'upload du package il est possible de le faire via le `setup.py` ou l'outil `twine`
  - Créer un job sur jenkins appelé Build
  - Ce job doit se déclencher en cas de succès du job "Test"
  - [Optionnel] mettre en place des notifications sur la pull request sur Github (api de notification) (cf annexe)
- [Optionnel] Mise en place de metrics sur les jobs (temps, % de success / failure)
- Script de déploiement du code:
  - Sur le serveur de déploiement mettez en place le fichier de configuration pour installer les packages python  
[https://pypicloud.readthedocs.io/en/latest/topics/getting\\_started.html#searching-packages](https://pypicloud.readthedocs.io/en/latest/topics/getting_started.html#searching-packages)
  - Appli web (cas basique inspiré de ça par exemple  
<http://acroca.com/blog/2014/07/17/zero-downtime-deployment-of-any-web-app-with-docker-and-nginx.html>).

Ce script de déploiement va déployer et démarrer une application en parallèle de sa version N-1, il devra lancer des health check afin de vérifier si tout l'application est prête à tourner si c'est le cas mettre à jour la conf nginx, reload le nginx afin d'éviter le downtime (et de faire un système de connexion draining) pour installer le package vous allez utiliser pip

  - Test de sante de l'application un curl sur /ping

A ce moment là on doit avoir un pipeline de delivery qui peut recevoir des updates du SCM et tester correctement le code, si tout est OK stock l'artefact puis le déployer

Tout ce qui est lié aux notifications et métriques sont optionnelles

Optionnel:

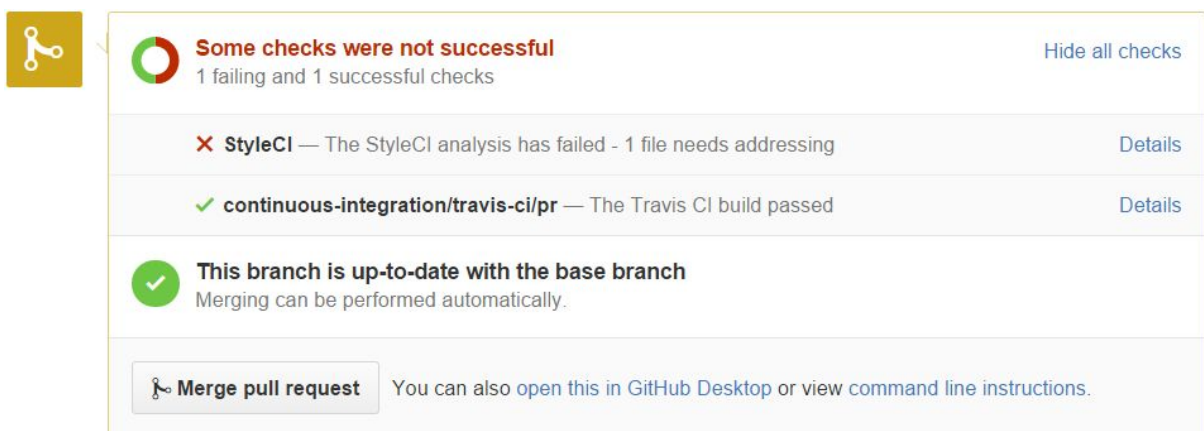
- Mettre en place un plugin pour afficher un pipeline qui représente l'enchaînement des jobs

Lexique:



- Continuous Integration (CI): est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée
- Continuous Delivery: est une approche en génie logiciel dans laquelle les équipes produisent du code dans des cycle court afin d'assurer la possibilité de livrer à n'importe quel moment
- Continuous Deployment: C'est comme le continuous delivery sauf qu'ici tout est automatisé et le déploiement vise la production. C'est à dire lorsque c'est mergé dans master le code part en production directement de façon automatisé
- Production: C'est l'environnement live où les client utilise votre service
- Staging: C'est l'environnement de avant la production aka pré-production, il sert de validation avant d'aller en production. Il faut qu'il soit iso production.
- SCM: source code management, ça représente les outils de gestion et centralisation de code par exemple: git
- Github: c'est un service SaaS qui propose du repository git infogéré
- Zero Downtime Deployment: permet de déployer une nouvelle version d'un système sans interrompre le bon fonctionnement du service.
- Pip: c'est le package manager de python


Annexe


Notification github:




The image shows a GitHub notification interface. At the top left is the GitHub logo. To its right is a red circle with a white 'X' icon, followed by the text "Some checks were not successful" and "1 failing and 1 successful checks". A link "Hide all checks" is on the right. Below this are two check items: "StyleCI" with a red 'X' icon and "continuous-integration/travis-ci/pr" with a green checkmark icon. At the bottom is a green checkmark icon followed by the text "This branch is up-to-date with the base branch" and "Merging can be performed automatically.". At the very bottom is a button "Merge pull request" and a link "You can also open this in GitHub Desktop or view command line instructions."

  **Some checks were not successful** [Hide all checks](#)  
1 failing and 1 successful checks

 **StyleCI** — The StyleCI analysis has failed - 1 file needs addressing [Details](#)

 **continuous-integration/travis-ci/pr** — The Travis CI build passed [Details](#)

 **This branch is up-to-date with the base branch**  
Merging can be performed automatically.

[Merge pull request](#) You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

