https://realpython.com/blog/python/docker-in-action-fitter-happier-more-productive/

https://github.com/realpython/fitter-happier-docker

**DOCKER IN ACTION : Integration Continue avec CircleCI**

Nous allons installé notre environnement local, en buildant une image à partir d'un Dockerfile puis créer une instance de l'image à partir du conteneur . Nous attacherons le tout avec Docker Compose pour builder et connecter différents conteneurs à la fois pour l'application et le processus Flask Redis .

**Sommaire**

1. **Prérequis**

**Installation d'une VM ubuntu 14.04 avec Virtualbox**

**ssh**

**docker**

1. **Docker**
   1. **Dockerfile**
   2. **docker-compose**
   3. **Build et exécution**
2. **GIT et GitHub**
3. **CircleCI**

**1 PREREQUIS**

**Avoir un compte sur :**

- github

- circleCI

- docker hub

**Installation d'une VM ubuntu 14.04 sur virtualbox**

Configurer ssh puis activer

$ sudo apt-get install ssh

$ sudo service ssh start

Mettre à jour les paquets

$ sudo apt-get update

Installer Docker (version la plus récente)

$ wget -qO- https://get.docker.com/ | sh

Vérifier que Docker est bien installé

$ sudo docker run hello-world

$ docker images

Ajouter un group "docker" et y ajouter votre user

$ sudo usermod -aG docker <user>

Rebooter puis vérifier que vous pouvez utiliser Docker sans "sudo"

$ docker run hello-world

Récupérer les fichiers du projet

git clone https://github.com/realpython/fitter-happier-docker.git

Renommer le répertoire

mv fitter-happier-docker circleci

**2 DOCKER**

**DOCKERFILE**

Le répertoire où sera le Dockerfile a été ajouté :

/home/user/circleci/web

# Dockerfile :

$ nano /home/user/circleci/web/Dockerfile

# start with a base image

FROM ubuntu:14.10

# install dependencies

RUN apt-get -y update

RUN apt-get install -y python python-dev python-pip python-psycopg2

RUN apt-get install -y nginx supervisor

# add requirements.txt and install

ADD requirements.txt /code/requirements.txt

RUN pip install -r /code/requirements.txt

# set working diretory

WORKDIR /code

**DOCKER COMPOSE**

L’objectif de “docker-compose” est de permettre d’exécuter et de mettre en relation les différentes images dont notre application a besoin en une seule commande. Un fichier docker-compose.yml doit être créé. Chaque container doit y être décrit : image, commande de lancement, volumes, ports…

Installer Docker Compose

$ curl -L https://github.com/docker/compose/releases/download/1.2.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

$ chmod +x /usr/local/bin/docker-compose

Tester l'installation de Compose

$ docker-compose --version

$ sudo nano docker-compose.yml

web:

build: web

volumes:

- web:/code

ports:

- "80:5000"

links:

- redis

command: python app.py

redis:

image: redis:2.8.19

ports:

- "6379:6379"

Ici, nous ajoutons les services qui composent notre pile :

1. **web** : tout d'abord, nous buildons l'image à partir du répertoire "web" et puis on monte ce répertoire dans le répertoire "code" dans le conteneur Docker . L'application Flask est lancé par la commande python app.py. Cela expose le port 5000 sur le conteneur, qui est transmise au port 80 de l'environnement hôte (port hôte:port conteneur)

2. **redis** : ensuite, le service Redis est buildé à partir de l'image "Redis" du Docker Hub. Port 6379 est exposé et transmis.

Noter que le Dockerfile et dans le répertoire "web". Ce fichier est utilisé pour builder notre image, à partir d'une base ubuntu, les dépendances requises sont installées et l'appli buildée

**Build et exécution**

Dorénavant, avec une simple commande on peut builder une image et lancer le conteneur :

$ docker-compose up

*Erreur " client and server don't have same version (client : 1.15, server: 1.12)" ==> docker upgrade*

*$ wget -qO- https://get.docker.io/gpg | sudo apt-key add -*

*$ sudo sh -c "echo deb http://get.docker.io/ubuntu docker main > /etc/apt/sources.list.d/docker.list"*

*$ sudo apt-get update*

*$ sudo apt-get install lxc-docker*

Cette commande a buildé une image de notre application Flask, récupérer l'image Redis, puis a tout démarré.

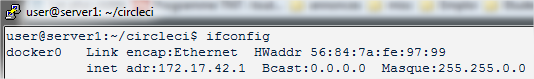
Docker Compose lance tous les conteneurs en même temps et en parallèle. Chaque conteneur a un nom unique et dans la pile trace/log, chaque processus a son code couleur.

Pour lancer docker-compose en mode "démon" (en arrière plan) :

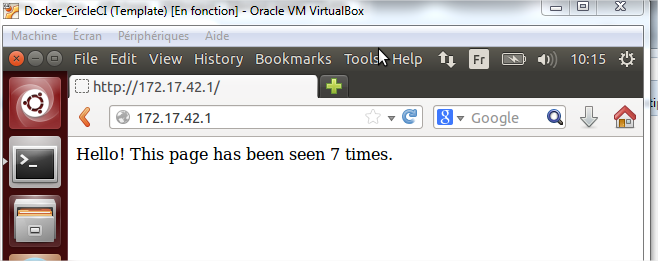
$ docker-compose up -d

**Test**

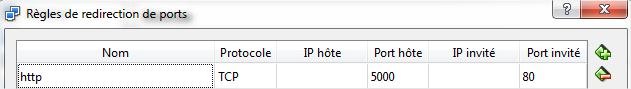
Repérer l'adresse IP affecter au conteneur (par ex. ifconfig - docker0)



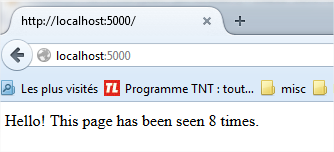
Dans votre navigateur de votre VM, saisir l'adresse IP docker : 172.17.42.1



Pour se connecter à partir du navigateur de la machine hôte, dans virtualbox, pour notre VM, faire une redirection de port : 5000 à 80.

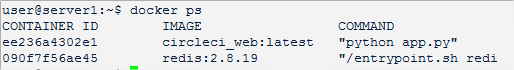
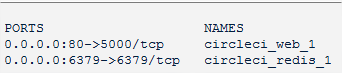


==> http://localhost:5000/

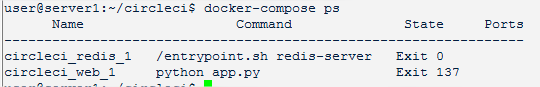


Afficher les processus en cours d'exécution :

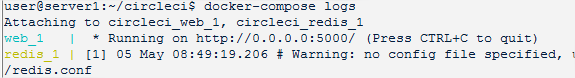
$ docker ps

$ docker-compose ps



Les 2 processus tournent sur des conteneurs différents, connectés via Docker compose :



Arrêter docker compose

$ docker-compose stop

J'indique les étapes ci-dessous pour ajouter une image, mais pour l'intégration continue, on ne peut pas mettre à jour directement une image dans le Docker Hub : Docker Hub récupère les modifications à partir de notre repo sur github et créé lui-même l'image.

Créer une image

$ cd /web # aller dans le répertoire qui contient le Dockerfile (/home/user/circleci/web)

$ docker build -t="christiprez/docker\_circleci" .

Vérifier que l'image a été buildée

$ docker images



$ docker push christiprez/docker\_circleci

Nous avons installer notre environnement local, en détaillant le processus de base de la construction d' une image d'un Dockerfile puis créer une instance de l'image (un conteneur) . Nous avons attaché le tout avec Docker Compose pour builder et connecter différents conteneurs à la fois pour l'application et le processus Flask Redis. Maintenant nous allons mettre en oeuvre l'intégration continue avec Github et CircleCI.

**3 GIT et Github**

http://git-scm.com/book/fr/v1/D%C3%A9marrage-rapide-Installation-de-Git

Nous allons committer puis pousser sur Github pour générer un nouveau build sur Docher Hub.

**1°) Installer git sur son serveur ubuntu**

$ apt-get update

$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev

$ apt-get install git

$ wget http://pkgs.fedoraproject.org/repo/pkgs/git/git-2.3.6.tar.gz/c29ac45bde67545b7e7941903000f6d3/git-2.3.6.tar.gz

$ tar -zxf git-2.3.6.tar.gz

$ cd git-2.3.6

$ make prefix=/usr/local all

$ sudo make prefix=/usr/local install

# Nom d’utilisateur : dire à git votre nom, pour qu'il étiquette les commits

$ git config --global user.name "christiprez"

# E-mail : Git sauvegarde l'adresse e-mail à l’intérieur des commits produits.

# L’adresse e-mail est utilisée pour associer les commits au compte GitHub.

$ git config --global user.email "christelleangus@yahoo.fr"

# Vérifier les paramètres saisis

$ git config --list

**2°) Démarrer un dépôt Git**

Vous pouvez principalement démarrer un dépôt Git de deux manières. La première consiste à prendre un projet ou un répertoire existant et à l'importer dans Git. La seconde consiste à cloner un dépôt Git existant sur un autre serveur.

Initialisation d'un dépôt Git dans un répertoire existant

Si vous commencez à suivre un projet existant dans Git, vous n'avez qu'à vous positionner dans le répertoire du projet et saisir :

$ git init

Cela crée un nouveau sous-répertoire nommé .git qui contient tous les fichiers nécessaires au dépôt — un squelette de dépôt Git. Pour l'instant, aucun fichier n'est encore versionné.

Si vous souhaitez commencer à suivre les versions des fichiers existants (contrairement à un répertoire vide), vous devriez probablement commencer par indexer ces fichiers et faire une validation initiale. Vous pouvez réaliser ceci avec une poignée de commandes git add qui spécifient les fichiers que vous souhaitez suivre, suivie d'une validation :

$ git add <fichier>

$ git commit –m 'version initiale du projet'

Nous allons passer en revue ce que ces commandes font dans une petite minute. Pour l'instant, vous avez un dépôt Git avec des fichiers sous gestion de version et une validation initiale.

Cloner un dépôt existant : https://github.com/amowu/hello-ci-workflow.git

Si vous souhaitez obtenir une copie d'un dépôt Git existant — par exemple, un projet auquel vous aimeriez contribuer — la commande dont vous avez besoin s'appelle **git clone**.

**3°) Créer un dépôt sur github**

Cliquez sur “Create a free account” et suivez les instructions pour créer votre compte

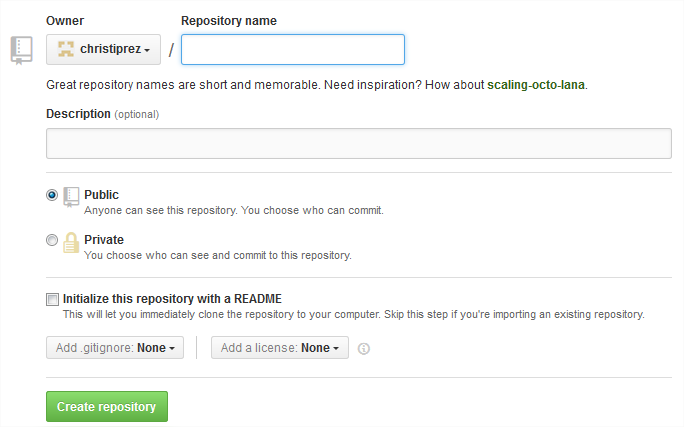
Créer un compte GitHub sur <https://github.com/plans>

Créer un nouveau dépôt ou repository :

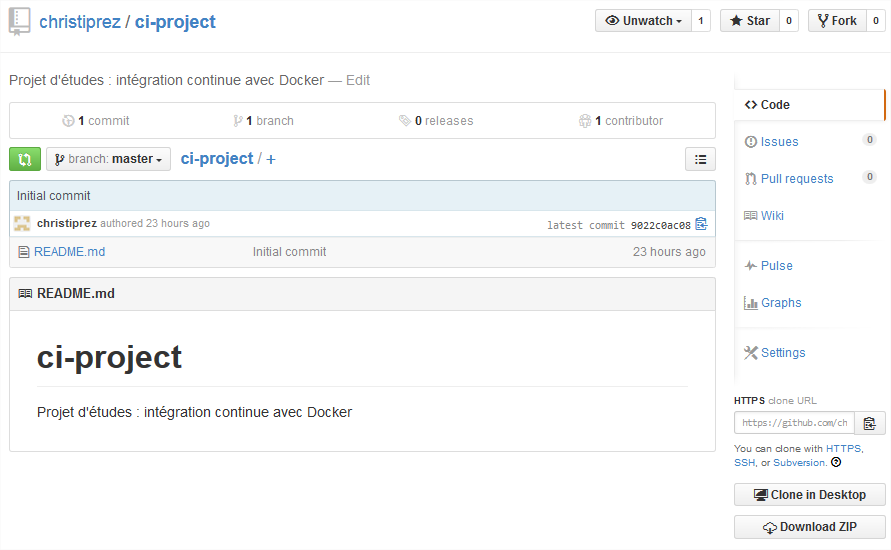
* Nous allons maintenant créer un nouveau dépôt. Cliquez sur “Create a New Repo” dans la section à droite en haut du site.

image

* Donnez un nom et une description à votre projet. Par défaut vous aurez un dépôt public et le projet sera vide à sa création. Nous pouvons y placer un fichier readme, en cochant la case “Initialize this project with a README”.



* Maintenant que le dépôt est créé, vous pouvez le cloner en local, sur votre ordinateur. Lancez le programme GitHub que vous venez d’installer. Vous allez ajouter les informations GITHUB pour que le programme puisse accéder à votre compte.



$ git add <fichier>

$ git add <dossier>

$ git status

$ git commit

$ git commit -m "Ajout Lisez-moi.txt"

$ git remote add origin https://github.com/christiprez/circleci.git

$ git remote -v

$ git push

$ git pull https://github.com/christiprez/circleci.git

$ git remote add origin https://github.com/christiprez/circleci.git

$ git push

**4 CircleCI**

Il est bon de savoir que si un commit provoque une régression, Docker Hub l'identifiera, mais puisque ceci est la dernière ligne de défense avant de déployer (soit la mise à jour ou en production) idéalement on veut identifier les régressions avant générer un nouveau build sur Docker Hub. De plus, on veut aussi exécuter les tests unitaires et d'intégration d'un véritable serveur d'intégration continue - qui est exactement où CircleCI entre en jeu.

S'inscrire à CircleCI avec son compte Github, puis ajouter le repo Github pour créer un nouveau projet. Cela ajoutera automatiquement un point Web d'accueil logiciel (webhook) au repo, de telle sorte que chaque fois que vous poussez vers Github, un nouveau build est lancé. Les points Web d'accueil logiciel sont attachés a vos registres et vous permettent de déclencher un évènement quand une image ou sa mise a jour est poussée dans le registre. On reçoit un email à chaque fois que le hook est ajouté.

Ensuite, on doit ajouter un fichier de configuration au répertoire racine, de sorte que CircleCI puis créer le build correctement.

**circle.yml**

Ajouter les commandes de build :

machine:

services:

- docker

dependencies:

override:

- pip install -r requirements.txt

test:

override:

- docker-compose run -d --no-deps web

- python web/tests.py

==> on créé une nouvelle image, lance le conteneur puis on lance les tests unitaires.

Ici, on a utilisé la commande docker-compose run -d --no-deps web , pour lancer le processus web, au lieu de docker-compose up . Ceci est parce que CircleCI a déjà Redis en cours et donc disponible pour les tests. Donc, on a juste besoin de lancer le processus web.

Avec le fichier circle.yml créé, pousser les changements sur Github pour déclencher un nouveau build. Rappel: ce qui déclenchera aussi un nouveau build sur Docker Hub.

**Démonstration**

**Créer une branche de fonction**

$ git checkout -b circleci-test master

Switched to a new branch 'circleci-test'

**Mettre à jour l'app**

Ajouter dans "/home/user/circleci/web/tests.py" : self.assertNotEqual(four, 60)

$ nano tests.py

**Lancer un pull**

$ git add web/tests.py

$ git commit -m "circleci-test"

$ git push origin circleci-test

Même avant avoir créer la demande de pull, CircleCI commence à lancer le build.

Lancer la demande de pull, puis lorsque les tests sont passées avec succès sur CircleCI, faire le merge. Une fois mergée, le build est lancé sur le Docker Hub.

**Réusinage du workflow (refactoring)**

1. Ouvrez votre repository sur Docker Hub, puis Settings et sélectionner Automated Build
2. Décochez la case Actif: “When active we will build when new pushes occur”.
3. Enregistrer.
4. Cliquez sur Build Triggers dans Settings
5. Changer le statut à "on"
6. Copiez l'exemple de commande curl :

$ curl --data "build=true" -X POST https://registry.hub.docker.com/u/mjhea0/fitter-happier-docker/trigger/84957124-2b85-410d-b602-b48193853b66/

Modifier le fichier *circle.yml* en ajoutant :

deployment:

hub:

branch: master

commands:

- $DEPLOY

Ici, nous tirons la variable $DEPLOY après avoir merger avec le maser et passer les tests. Nous allons ajouter la valeur réelle de cette variable comme une variable d'environnement sur CircleCI:

1. Ouvrir *Project Settings*, puis cliquer *Environment variables*.
2. Ajouter une nouvelle variable “DEPLOY” et comme valeur, y affecter la commande curl ci-dessus

Ouvrir une nouvelle demande de pull, puis une fois les tests passés sur Cercle CI, merger sur master. Un autre build est déclenché. Puis, une fois les tests passés, un nouveau build est déclenchée sur Docker Hub via la commande curl.

**CONCLUSION**

Donc, voici la mise en place d'un environnement local couplé avec l'intégration continue via [CircleCI](https://translate.googleusercontent.com/translate_c?depth=1&hl=fr&rurl=translate.google.com&sl=auto&tl=fr&u=https://circleci.com/&usg=ALkJrhjl8RYAlNhzpqvpH1aZY6_giIIlGw) (étapes 1 à 6):

1. Coder localement sur une branche de fonction
2. Ouvrir une demande de pull sur Github sur la branche master
3. Exécuter des tests automatisés sur le conteneur Docker
4. Si les tests réussissent, fusionner manuellement la demande de pull sur le master
5. Une fois fusionné, les tests automatisés fonctionnent
6. Si le second tour de tests passent, un build est lancé sur Docker Hub

**ANNEXE**

SOURCES

http://christopheducamp.com/2013/12/10/installer-git/

http://learnosm.org/fr/advanced/github-sharing/

http://doc.ubuntu-fr.org/git

**Commandes Spécifiques Git**

Le fait que Git ait été conçu avec un gros projet comme Linux, il existe un paquet de commandes Git. Néanmoins, pour utiliser les basiques de Git, vous n’aurez besoin de connaître que quelques termes. Ils commencent tous de la même façon avec le mot “git”.

git init : Initialise un nouveau dépôt Git. Jusqu’à ce que vous exécutiez cette commande dans un dépôt ou répertoire, c’est simplement un répertoire normal. Seulement après avoir entré cette commande, il accepte les commandes Git qui suivent.

git config : raccourci de “configurer,” ceci est tout particulièrement utile quand vous paramétrez Git pour la première fois.

git help : Oublié une commande ? Saisissez-ça dans la ligne de commande pour ramener les 21 commandes les plus courues de Git. Vous pouvez aussi être plus spécifique et saisir “git help init” ou tout autre terme pour voir comment utiliser et configurer une commande spécifique git.

git status : Vérifie le statut de votre repository. Voir quels fichiers sont à l’intérieur, quels sont les changements ayant besoin d’être gardés en mémoire, et sur quelle branche du repository vous êtes en train de travailler.

git add : Ceci n’ajoute *pas* de nouveaux fichiers dans votre dépôt. Au lieu de cela, cela porte de nouveaux fichiers à l’attention de Git. Après que vous ayez ajouté des fichiers, ils sont inclus dans les “instantanés” du dépôt Git.

git commit : la commande la plus importante de Git. Après avoir fait toute sorte de modification, vous saisissez ça afin de prendre un “instantané” du dépôt. Généralement cela s’écrit sous la forme git commit -m “Message ici“. Le -m indique que la section suivante de la commande devrait être lue comme un message.

git branch : Vous travaillez avec plusieurs collaborateurs et voulez produire des modifications de votre côté ? Cette commande vous permet de construire une nouvelle branche, ou une timeline de commits, de modifications et d’ajouts de fichiers qui sont complètement les vôtres. Votre titre va après la commande. Si vous vouliez créer une nouvelle branche appelée “chats”, vous saisiriez git branch chats.

git checkout : Permet littéralement de rapatrier un dépôt dans lequel vous n’êtes pas. C’est une commande de navigation qui vous laisse migrer vers le répertoire que vous voulez rapatrier. Vous pouvez utiliser cette commande sous la forme git checkout master pour rapatrier la branche master, ou git checkout chats pour rapatrier une autre branche.

git merge : Quand vous avez terminé de travailler sur une branche, vous pouvez fusionner vos modifications vers la branche master, qui est visible pour tous les collaborateurs. git merge cats prendrait toutes les modifications que vous avez faites sur la branche “cats” et les ajoutera à la la branche master.

git push : Si vous travaillez sur votre ordinateur local, et voulez que vos commits soient visibles aussi sur Github, vous “push”ez les modifications vers Github avec cette commande.

git pull : Si vous travaillez sur votre ordinateur local, et su vous voulez la version la plus à jour de votre repository pour travailler dessus, vous “pull”ez (tirez) les modifications provenant de Github avec cette commande.

**Installation serveur git**

(https://www.howtoforge.com/how-to-run-your-own-git-server-with-gitlabhq-on-ubuntu-14.04)

1°) Installer Ruby, Python et Bundler Gem

apt-get update

apt-get install -y build-essential zlib1g-dev libyaml-dev libssl-dev libgdbm-dev libreadline-dev libncurses5-dev libffi-dev curl git-core openssh-server redis-server checkinstall libxml2-dev libxslt-dev libcurl4-openssl-dev libicu-dev

mkdir /tmp/ruby && cd /tmp/ruby

wget http://ftp.ruby-lang.org/pub/ruby/2.1/ruby-2.1.2.tar.gz

tar zxvf ruby-2.1.2.tar.gz

cd ruby-2.1.2

./configure

make

make install

gem install bundler --no-ri --no-rdoc

# ajouter un utilisateur git pour Gitlab

adduser --disabled-login --gecos 'GitLab' git

2°) Installation du shell Gitlab

cd /home/git  
sudo -u git -H git clone https://github.com/gitlabhq/gitlab-shell.git  
cd gitlab-shell  
sudo -u git -H git checkout v1.7.0  
sudo -u git -H cp config.yml.example config.yml

nano config.yml

modifier la valeur de gitlab\_url: "http://localhost/"

en : gitlab\_url: "http://server1.example.com"

sudo -u git -H ./bin/install

3°) Installation de Mariadb pour Gitlab

apt-get install mariadb-server mariadb-client libmariadbclient-dev

*user/mot de passe : root/* ***mariadbpassword***

mysql -u root -p

*database=gitlabdb user=gitlabuser and password=gitlabuserpassword:*

# dans Mariadb :

*CREATE DATABASE gitlabdb DEFAULT CHARACTER SET utf8 COLLATE utf8\_unicode\_ci;  
GRANT SELECT, LOCK TABLES, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER ON gitlabdb.\* TO 'gitlabuser'@'localhost' IDENTIFIED BY 'gitlabuserpassword';  
flush privileges;  
quit*

cd /home/git  
sudo -u git -H git clone https://github.com/gitlabhq/gitlabhq.git gitlab  
cd /home/git/gitlab  
sudo -u git -H git checkout 6-0-stable  
sudo -u git -H cp config/gitlab.yml.example config/gitlab.yml

sudo -u git -H nano config/gitlab.yml

*## GitLab settings*

*# gitlab:*

*# ## Web server settings*

*# host: localhost*

*# port: 80*

*# https: false*

*gitlab:*

*## Web server settings*

*host: server1.example.com*

*port: 80*

*https: false*

# permissions

cd /home/git/gitlab  
chown -R git log/  
chown -R git tmp/  
chmod -R u+rwX  log/  
chmod -R u+rwX tmp/  
sudo -u git -H mkdir /home/git/gitlab-satellites  
sudo -u git -H mkdir tmp/pids/  
sudo -u git -H mkdir tmp/sockets/  
chmod -R u+rwX tmp/pids/  
chmod -R u+rwX tmp/sockets/  
sudo -u git -H mkdir public/uploads  
chmod -R u+rwX  public/uploads  
sudo -u git -H cp config/unicorn.rb.example config/unicorn.rb  
sudo -u git -H git config --global user.name "GitLab"  
sudo -u git -H git config --global user.email "gitlab@localhost"  
sudo -u git -H git config --global core.autocrlf input  
sudo -u git cp config/database.yml.mysql config/database.yml

# indiquer à Gitlab le user du gitlab Mariadb : config/database.yml

sudo -u git -H vi config/database.yml

*#production:*

*# adapter: mysql2*

*# encoding: utf8*

*# reconnect: false*

*# database: gitlabhq\_production*

*# pool: 10*

*# username: root*

*# password: "secure password"*

*production:*

*adapter: mysql2*

*encoding: utf8*

*reconnect: false*

*database: gitlabdb*

*pool: 10*

*username: gitlabuser*

*password: "gitlabuserpassword"*

sudo -u git -H chmod o-rwx config/database.yml

# installer quelques gems supplémentaires :

cd /home/git/gitlab  
gem install charlock\_holmes --version '0.6.9.4'

# télécharger les paquets et les installer

wget https://downloads-packages.s3.amazonaws.com/ubuntu-14.04/gitlab\_7.1.1-omnibus-1\_amd64.deb  
apt-get install openssh-server  
apt-get install postfix # utiliser <tab> pour sélectionner. Select 'Internet Site', using sendmail instead also works, exim has problems

dpkg -i gitlab\_7.1.1-omnibus-1\_amd64.deb

nano /etc/gitlab/gitlab.rb

*#external\_url 'http://gitlab.example.com'*

*external\_url 'http://server1.example.com'*

gitlab-ctl reconfigure

Navigateur : http:// 192.168.0.101/users/sign\_on

username=root

password=5iveL!fe (changer en "password")

SOURCES et LIENS

https://realpython.com/blog/python/docker-in-action-fitter-happier-more-productive/

https://www.howtoforge.com/how-to-run-your-own-git-server-with-gitlabhq-on-ubuntu-14.04

http://www.kf-interactive.com/blog/roll-your-own-docker-registry-with-docker-compose-supervisor-and-nginx/

https://linuxmeerkat.wordpress.com/tag/circleci/

http://docs.docker.com/compose/

http://www.disko.fr/reflexions/technique/introduction-docker/

Exemple:

http://anandmanisankar.com/posts/docker-container-nginx-node-redis-example/

DEFINITIONS

[Docker Compose](http://docs.docker.com/compose/) is an orchestration framework that handles the building and running of multiple services (via separate containers) using a simple .yml file. It makes it super easy to link services together running in different containers.