USINE LOGICIELLE

Setup VirtualBox

* Téléchargements: https://www.virtualbox.org/wiki/Downloads

Windows

* télécharger l'exécutable et exécuter le wizard d'installation avec les options par défaut

Mac

* télécharger le .dmg et l'exécuter

Linux (Debian Family)

* version du dépôt debian 6.1.16

Setup Vagrant

* Téléchargements: https://www.vagrantup.com/downloads

ubuntu

* https://linuxize.com/post/how-to-install-vagrant-on-ubuntu-20-04/

confirmation

* taper dans un terminal : `vagrant -v`

lancement de la vagrant box "myusine"

ajouter la box myusine à vagrant

* créer un dossier "usine"
* placer les deux parties du zip .001 et .002
* Windows:
    * installer 7-zip et dézipper le .001
* Linux:
    * `sudo apt-get istall unrar` ou 7-zip

* MacOS:
    * 7-zip ou the unarchiver (clic droit ouvrir avec sur le .001)
* ajouter la box myusine.box dans vagrant: `vagrant box add myusine myusine.box`
    * confirmation: **==> box: Successfully added box 'myusine' (v0) for 'virtualbox'!** ou `vagrant box list`

lancer une vm de type "myusine" dans virtualbox avec le Vagrantfile

* dans le fichier Vagrantfile:
    * trouver l'interface réseau sur laquelle on se connecte à internet (câble -> Ethernet, Wifi -> carte Wifi
    * Windows: gestionnaire de périphérique -> carte réseau -> propriétés détail -> copier
    * Windows: powershell -> ipconfig /all -> champ description de l'interface
    * mac et linux: `ip a` ou `ifconfig`
    * le nom de l'interface doit être copié dans la variable **int**
    * entrer une adresse ip disponible du sous réseau local
    * lire le masque de sous réseau : 255.255.255.0 => cidr = "24", 255.255.0.0 => cidr = "16"
    * savoir si une adresse est disponible = `ping 192.168.x.y` ne doit répondre
    * renseigner la variable range avec cette ip
    * lancer la vm dans le dossier usine où se trouve le Vagrantfile: `vagrant up`
    * si problèmes d'espace disque, dans virtualbox: section "outils" -> paramètres -> Général -> Dossier par défaut des machines -> renseigner un chemin sur un un support ayant > 65Go

connecter l'usine logicielle depuis le navigateur

* réslution dns entre le nom de domaine gitlab.myusine.fr et l'ip de la vm
    * Windows: éditer le fichier `C:\Windows\System32\drivers\etc\hosts`
        * ajouter la ligne `192.168.x.y gitlab.myusine.fr`
* Sous Unix (os x && Lixu(s)): éditer `sudo nano /etc/hosts`
* se connecter à gitlab.myusine.fr dans un navigateur.
* résoudre l'exception de sécurité => avancé => poursuivre (certificat SSL https auto signé)
* login: root, mdp: roottoor

création d'un dépôt local de code versionné par git

* setup dossier partagé /vagrant
    * la vm gitlab.myusine.fr possède un répertoire partagé avec l'hôte
    * le dossier /vagrant dans la vm correspond au dossier dans lequel se trouve le Vagrantfile
    * se connecter à la vm avec `vagrant ssh`
    * afficher le contenu du dossier /vagrant: `ls -al /vagrant`

création du dépôt

* créer un répertoire local dans `/home/vagrant` sur la vm: `mkdir /home/vagrant/local`
* entrer dans le dossier local
* initialiser un dépôt de code git: `git init`

extension remote SSH dans VSCode

* dans vscode installer l'extension remote ssh
* cliquer sur l'icône remote explorer
* appuyer sur le bouton +
* entrer `ssh@ip.de.la.vm`
* choisir un fichier de configuration ssh
    * "unix": /home/"utilisateur"/.ssh/config
    * windows: c:\users\"utilisateur"\.ssh\config
    * le code généré:
      ```
      Host 192.168.1.78
        Hostname 192.168.1.78
        User vagrant
      ```
    * sur l'hôte dans le dossier du vagrantfile: `vagrant ssh-config`
    * copier la ligne IdentityFile: `IdentityFile "C:/Users/Admin stagiaire.DESKTOP-8967908/.vagrant.d/insecure_private_key"`
    * la coller dans la fenêtre de droite de vscode
    ```
      Host 192.168.1.78
        Hostname 192.168.1.78
        User vagrant
        IdentityFile "C:/Users/Admin stagiaire.DESKTOP-8967908/.vagrant.d/insecure_private_key"`
    ```
    * clic droit sur la target ssh dans la fenêtre de gauche de vscode
    * connect to remote Host in current window
    * open Folder: `home/vagrant/local`

commandes de base avec git

* `git init`pour initialiser un dépôt dans un dossier
* création d'un contenu
* `git status` pour voir l'état des fichiers de la copie de travail
* `git add`pour ajouter un fichier dans la zone de préparation du commit, i.e la nouvelle version
* `git commit -m` pour enregistrer les modifications de la zone de préparation vers le dépôt
* `git config (--global) user.(name|email)  "..."` pour renseigner les metadonnées néessaires du commit
* `git log -p`: affiche l'historique des commits avec métadonnées et (-p) la diff par rapport au commit précédent
* `git branch (-l| -v)` affiche les branches et la branche courante
* `git branch -m "new_name" renomme la branche courante

connexion du dépôt local au dépôt gitlab

* créer une paire de clé privée publique: `ssh-keygen`

    * chemin /home/vagrant/.ssh/gitlab, pas de passphrase

* copier le contenu de la clé publique (`cat /home/vagrant/.ssh/gitlab.pub` dans les settings utilisateurs, section **SSH Keys**i
* dans le dépôt local, `git remote add origin git@gitlab.myusine.fr:root/myusine.git` pour ajouter le dépôt distant gitlab

* git push origin main pour pousser le code

* si la clé privée n'est pas reconnue:
    * éditer le fichier `/home/vagrant/.ssh/config`

```
Host gitlab.myusine.fr
 HostName gitlab.myusine.fr
 User git
 IdentityFile /home/vagrant/.ssh/gitlab
```

HELLO WORLD de l'integration contiue avec gitlab

* créer un fichier **.gitlab-ci.yml**

```
build:
  script:
    - echo "Build"

test:
  script:
    - echo "Test"

deploy:
  script:
    - echo "Deploy"
```

* pousser le fichier sur gitlab
    * `git add . && git commit -m "gitlab-ci" && git push origin main`

remarques sur l'exécution du gitlab-ci.yml

synchronicité

* par défaut les jobs sont exécuté en concurrence

* par défaut, le processus de gitlab-runner est monothread
* on peut rendre gitlab-runner multithread en configurant le paramètre **concurrent** dans le fichier **/etc/gitlab-runner/config.toml**
* on peut déclencher les jobs de façon manuelle dans le menu pipelines -> **Run Pipelines**

conditionnalité

* `git mv`: permet de renommer un fichier dans le système de fichier ET dans le dépôt git
* `git rm`: permet de suppromer un fichier dans le système de fichier ET dans le dépôt git

* jobs de départ pour la clé when

```
image: python:rc-slim-buster

stages:
  - builds
  - tests

build:
  stage: builds
  script:
    - echo "build"

build2:
  stage: builds
  script:
    - echo "build2"

test:
  stage: tests
  needs: [build]
  script:
    - echo "Test"

test2:
  stage: tests
  script:
    - echo "Test"
```

* `git branch "nom_de_la_branche": création d'une nouvelle branche
* `git checkout "nom_de_la_branche": basculement sur une autre branche
* `git branch -d "nom de la branche": suppression d'une branche depuis une autre branche
* `git checkout -b "nom_de_la_branche" : création et basculement sur une nouvelle branche

Merge Request

* fusion de branches sur le dépôt de référence gitlab
* menu Merge Request du projet
* détection des push sur les branche => "Create Merge Request"
* configuration de la MR (titre, description, responsable de MR, réviseurs)
* "submit MR" => les réviseurs reçoivent un email
* Les réviseurs analysent le code et décident ou non de fusionner
* "Merge" => fusionne les branches sur gitlab et déclenche un pipeline sur la branche main

* sur le dépôt local on récupère la fusion avec: `git pull origin main` dans la branche main

* sur le dépôt local, on supprime la branche feature1: `git branch -d feature1`

remarque sur la clé cache

* le cache géré localement se retrouve dans un volume docker:
    * `docker volume ls` pour voir les cache
    * `docker volume prune`pour suprimer les caches
    * `docker volume rm nom_du_cache` pour supprimer un cache en particulier

* exemple d'utilisation de la clé cache:

```
build:
  stage: builds
  script:
    - mkdir cache
    - echo "cache" > cache/build_cache
  # création d'un volume docker avec
  # le dossier cache/ comme contenu
  cache:
    key: build_cache
    paths:
      - cache/
    policy: push

test:
  stage: tests
  script:
    - cat cache/build_cache
  # utilisation du cache précédent
  cache:
    key: build_cache
    untracked: true
    policy: pull
```

```

## premier job d'intégration continue

### setup projet:

* installer le contenu de projet.zip dans le dossier /home/vagrant/local de la vm (ou dans le dépôt local sur windows)
    * déposer le zip dans le dossier du Vagrantfile qui est partagé avec la vm
    * `sudo apt update && sudo apt install -y unzip`pour installer le paquet unzip sur la vm
    * `sudo unzip /vagrant/projet.zip -d /home/vagrant/local` pour dézipper dans le dépôt
    * `sudo chown -R vagrant:vagrant /home/vagrant/local` pour rétablir les propriétaires

### environnement virtuel

* après installation du projet, python3 est disponible sur la vm
* il faut installer le gestionnaire de paquet python qui s'appelle pip3
  * `sudo apt install -y python3-pip`
* il faut installer le paquet permettant d'utiliser les environnements virtuels
  * `sudo apt install -y python3-venv`
* on va créer un environnement virtuel dans le dépôt local sous la forme d'un dossier **venv**
    * Linux: `python3 -m venv venv`
    * Windows: `py -m venv venv`
* on va activer l'environnement virtuel:
    * Linux: `source venv/bin/activate`
    * windows: `.\venv\Scripts\Activate.ps1`

### installation des dépendances

* il faut installer le paquet **bottle** dans l'environnement virtuel
  * `pip3 install bottle`
  * `pip install bottle`

### lancer l'application dans l'environnement virtuel

* `renseigner l'ip de la vm dans le fichier app.py dans le paramètre host`
* lancer avec `python3 app.py`

### automatiser l'installation de l'environnement virtuel

* position de départ:
    * les sources du projet poussées dans gitlab
    * le fichier gitlab-ci.yml initialisé avec :

```
workflow:
```
```

```
  rules:
    # le pipeline se déclenche dès qu'une règle est vérifiée
    # OU logique
    - if: $CI_PIPELINE_SOURCE == "push"
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"

image: python:rc-slim-buster

create_env:
    stage: init
    script:
        - echo "create venv"
```

* générer les dépendances dans un fichier requirements.txt (cf slides)
    * sous linux, virer la dépendance pkg-resources
    * `pip3 freeze | grep -v pkg-resources > requirements.txt`

* pousser le sources sur gitlab
    * mettre à jour le .gitignore

* écrire le job create_env
    1. installer le paquet python3-venv
    2. créer l'environnement virtuel dans un dossier venv
    3. activer l'environnement virtuel
    4. installer les dépendances du projet
    5. mettre en cache le dossier venv
    6. tester le cache dans un job test_env

```
stages:
  - init
  - tests

create_env:
  stage: init
  before_script:
    # sur un conteneur, sudo n'est pas présent
    # par défaut l'utilisateur est root
    # par défaut le cache apt est absent
    # on utilisera apt-get au lieu d'apt
    # - apt-get update
    # - apt-get install -y python3-venv
    - python3 -m venv venv
    - source venv/bin/activate
  script:
```

```
    - pip3 install -r requirements.txt
  cache:
    key: venv
    paths:
      - venv/
    policy: push

test_env:
  stage: tests
  script:
    - test -e venv
  cache:
    key: venv
    untracked: true
    policy: pull
```

* le job create_env n'a pas besoin d'être exécuté à chaque lancement de pipeline, puisque le cache est persistent, tantque l'environnement virtuel n'évolue

* on n'exécute le job create_env que si requirements.txt a changé

* on peut également décider d'exécuter le job manuellement

```
variables:
    CREATE_CACHE: "off"
...
rules:
    - changes:
        - requirements.txt
    - if: $CREATE_CACHE == "on"
```

tests unitaires

* méthodologie BDD: utilisation du logiciel cucumber avec le language Gherkin

* exemple gherkin

```
Feature: claculate average number
Scenario Outline: test avg on a list of integers
Given we have a list of ints <liste>
When we calculate avg
Then avg is <result>
```

Examples: Listes
| liste | result |
| [1, 2, 3]| 2.0 |
```

* exemple de statisfaction de l'expression de besoin

```
from behave import *
from test_assert import avg
import json

@given('we have a list of ints {liste}')
def step(context, liste):
context.liste = json.loads(liste)
assert isinstance(context.liste, list), "not a list"

@when('we calculate avg')
def step(context):
context.avg = avg(context.liste)

@then('avg is {result}')
def step(context, result):
assert context.avg == float(result), f"{context.avg}"
```

lancement de tests unitaires via unittest
  * `python3 -m unittest chemin_python` i, e package, subpackage, module, classe, méthode
  * `python3 -m unittest discover: algorithme de découverte de tests par scrutation du dossier courant
  * exécuter une suite de test: `python3 -m unittest run_tests.py`

* les concepts fondamentaux du test:
   * le TestCase
   * les fixtures : ressources logicielle nécéssaires au test (objet instancié, connexion bdd ou api)
   * La TestSuite: selection de tests à exécuter
   * Le TestRunner: classe d'export du rapport de test
   * Le Mock : une classe qui simule le résultat d'un composant utilisé par le test sans être testé

* exemple naïf de job

```
units:
  stage: tests
  before_script:
    - source venv/bin/activate
  script:
```
```

```
    - python3 -m unittest run_tests.py
  cache:
    key: venv
    untracked: true
    policy: pull
```

remontée du rapport du test dans gitlab

* gitlab peut accepter un artefact au format xml Junit
* pour générer un tel rapport, on doit utiliser un paquet python spécifique: `pip3 install unittest-xml-reporting`
* on va utiliser la suite de test du fichier **coverage_tests.py** : `python3 coverage_tests.py`
* on regénère les dépendances:
    * il faut distinguer les dépendances projet des dépendances lié à l'intégration continue
    * `pip3 freeze | grep -E -v "pkg-resources|bottle" > requirements-ci.txt
    * `pip freeze | ?{$_ -notmatch "bottle"} > requirements-ci.txt` dans windows

* on ajuste le job **units** en exécutant **coverage_tests.py** et en utilisant la clé **artifacts:reports;junit:**
* on ajuste le job **create_env** en tenant compte des deux fichiers de dépendances requirements.txt et requirements-ci.txt

```
create_env:
  stage: init
  before_script:
    # sur un conteneur, sudo n'est pas présent
    # par défaut l'utilisateur est root
    # par défaut le cache apt est absent
    # on utilisera apt-get au lieu d'apt
    - python3 -m venv venv
    - source venv/bin/activate
  script:
    - pip3 install -r requirements.txt
    - pip3 install -r requirements-ci.txt
  cache:
    key: venv
    paths:
      - venv/
    policy: push
  rules:
    - changes:
      - "requirements*.txt"
    - if: $CREATE_CACHE == "on"
```

```
units:
  stage: tests
  before_script:
    - source venv/bin/activate
  script:
    - python3 coverage_tests.py
  cache:
    key: venv
    untracked: true
    policy: pull
  artifacts:
    reports:
      # les chemins utilisant des wildcards "*" ou "**"
      # doivent être entourés de doubles quotes ""
      junit: "reports/TEST-*.xml"
```

couverture de code

* instalation du package **coverage** : `pip3 install coverage`
* régénération des dépendance CI: `pip3 freeze | ... > requirements-ci.txt`

* lancement des tests unitaires à travers le module coverage:
    * `coverage run coverage_tests.py`

* affichage du rapport de couverture
    * `coverage report -m`

* retravailler la commande coverage run pour sélectionner uniquement les classes account, client, et factories du package bank: https://coverage.readthedocs.io/en/coverage-5.5/cmd.html
    * `coverage run --include="bank/*.py" --omit="**/__init__.py,**/tests/*.py" coverage_tests.py`

* adapter le job units pour prendre en compte la couverture de code

```
script:
    - >
      coverage run
      --include="bank/*.py"
      --omit="**/__init__.py,**/tests/*.py"
      coverage_tests.py
    - coverage report -m
```

* on peut configurer dans les réglages CI/CD du projet (section "General Pipelines") une expression régulière qui va prélever la couverture TOTALE pour l'afficher dans l'interface des jobs

tests de bout en bout ou test end to end (E2E)

client selenium connecté au navigateur Firefox de la machine

* installer selenium dans le venv: `pip3 install selenium`
* regénérer les dépendances CI: `pip3 freeze ...`

gérer le driver de connection
* rendre accessible et exécutable le driver geckodriver dans la vm pour le client selenium
    * afficher le PATH linux: `echo $PATH`
    * ou le PATH Windows: `$Env:Path`
    * copier le geckodriver dans /usr/local/bin: `sudo cp geckodriver /usr/local/bin`
    * rendre geckodriver exécutable: `sudo chmod +x /usr/local/bin/geckodriver`

installer firefox

* `sudo add-apt-repository ppa:mozillateam/ppa`
* `sudo apt update && sudo apt install -y firefox-esr`
* il faut configurer le client sélénium pour que le driver connaisse l'exécutable firefox
    * pour renseigner le chemin vers l'exécutable firefox
    * pour passer le navigateur en mode "HeadLess" dans un environnement sans interface graphique
```
import unittest
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.firefox.options import Options

class PythonOrgSearch(unittest.TestCase):

    def setUp(self):
        options = Options()
        options.headless = True
        options.binary_location = '/usr/bin/firefox-esr'
        self.driver = webdriver.Firefox(options=options)
```

* on exécute le script: `python3 e2e_firefox.py`

automatisation avec un serveur selénium comme service réseau

1. on passe à un client selenium instanciant une connexion à un serveur sélénium
2. on gère le driver (cf supra)
3. on va se doter d'un service réseau selenium sous la forme d'un conteneur docker selenium pour firefox au

moyen de la clé **services** de gitlab

* le job:

```
e2e:
  stage: tests
  services:
    - name: selenium/standalone-firefox:latest
      alias: gitlab_selenium_server
  before_script:
    - cp geckodriver /usr/local/bin
    - chmod +x /usr/local/bin/geckodriver
    - source venv/bin/activate
  script:
    - python3 e2e_remote.py
  cache:
    key: venv
    untracked: true
    policy: pull
```

* test du job: on désactive le job unit avec
```
rules:
    - when: never
```

optimisations du code et multithreading

* on peut factoriser l'utilisation du cache grâce aux alias et ancres YAML

```
.cache: &cache
  key: venv
  untracked: true
  policy: pull
```

et la clé cache des jobs utilisant ce cache: `cache: *cache`

* le cache est généré dans un volume qui dépend d'un thread
* OR, deux jobs en parallèle nécessitent deux threads
* DONC un des deux jobs seulement disposera du cache
* Correction: on utilise la clé **parrallel** pour exécuter le job create_env dans plusieurs threads
* Forcer la régénération du cache avec la variable CREATE_CACHE à "on"

Qualité - Analyse statique de Code

le serveur SonarQube

* lancement d'un conteneur docker sonarqube:
  `docker container run -d --name sonar --restart unless-stopped -p 9000:9000 sonarqube:lts`
    * -d: conteneur détaché du processus qui le lance
    * --name : étiquette dans docker
    * --restart : politique de redémarrage du conteneur en cas de crash
    * -p redirection de port sur l'hôte du conteneur

* vérifier le lancement: `docker ps`

* connection à la vm sur le port 9000 dans un navigateur

* login /mdp : admin / admin

* création du projet (clé projet et token)
* création du profile qualité associé au projet (profile python custom)
* création de seuils de validation de l'analyse associés au projets

Le job dans gitlab

* on utilise la commande sonar-scanner délivrée par le serveur
* on affine la commande pour n'analyser que les sources correspondant aux fichiers .py
* on va utiliser une image docker contenant le sonar-scanner: `sonarsource/sonar-scanner-cli:latest`
* on utilise l'ip de la vm depuis le conteneur d'intégration pour se connecter
  * `docker network ls` pour voir les réseaux docker
  * `docker network inspect bridge` pour voir la configuration du réseau par défaut
* on va cacher le token d'identification dans une variable gérée par l'interface gitlab
  * settings -> CI/CD -> Variables

```
sonar:
  image: sonarsource/sonar-scanner-cli:latest
  stage: quality
  script:
    - >
      sonar-scanner
      -Dsonar.projectKey=myusine
      -Dsonar.sources=.
      -Dsonar.inclusions=**/*.py
      -Dsonar.exclusions=**/*.css
      -Dsonar.host.url=http://172.17.0.1:9000
      -Dsonar.login=$SONAR_TOKEN
```

```

* on peut faire remonter le rapport de couverture de code dans sonarqube, si celui ci est en xml au format cobertura
    * génération du rapport de coverage xml : `coverage xml -o reports/coverage.xml`
    * prélèvement de l'artefact avec **artifacts:paths**
    * option sonar pour inclure le rapport: **-Dsonar.python.coverage.reportPaths**
    * options sonar pour résuir le champs de couverture: **-Dsonar.coverage.exclusions**

```
      -Dsonar.python.coverage.reportPaths=reports/coverage.xml
      -Dsonar.coverage.exclusions=**/__init__.py,**/tests/*.py,*.py
```

## déploiement via ansible

* on va déployer le code depuis un conteneur d'intégration continu vers la vm, sur le compte d'un utilisateur **ansibe**

*création  d'un utilisateur **ansible** sur la vm:
    * `sudo useradd` dans un script de création d'utilisateur
    * `sudo adduser ansible` pour le faire de façon interactive

* configurer le serveur ssh pour accepter les connexions par clés publiques/privées
    * `sudo nano /etc/ssh/sshd_config`
    * décommenter la ligne: #PubkeyAuthentication yes
    * recharger la configuratin du serveur: `sudo service sshd reload`

* création des clés privées / publiques pour autoriser les connections ssh sur l'utilisateur ansible
    * connexion sur le compte ansible: `su ansible -`, puis `cd /home/ansible`
    * `ssh-keygen`, chemin **/home/ansible/.ssh/ansible**, sans passphrase

* ajout du contenu de la clé publique dans le fichier .ssh/authorized_keys, de droits 600 (rw-------) qui contrôle la validité des clés privées envoyés par les clients de connexions ssh
    * `mv .ssh/ansible.pub .ssh/authorized_keys && chmod 600 .ssh/authorized_keys`

* transférer la clé privée sur le compte vagrant:
    * `exit` pour revenir sur le compte vagrant
    * `sudo mv /home/ansible/.ssh/ansible ~`

test de la connexion ssh depuis vagrant:
    * selon la norme ssh, le dossier contenant la clé privée doit être en 700 et la clé en 400
    * `sudo chown vagrant:vagrant ansible && sudo chmod 700 . && sudo chmod 400 ansible`
    *`ssh -i ansible ansible@127.0.0.1`

on va configurer la connexion depuis gitlab dans un job:
```

    * on va utiliser l'image docker **ansible/ansible:default**
    * il faut placer le contenu de la clé privée dans une variable cachée (Settings -> CI/CD -> Variables)
    * il faut s'assurer que le dossier a des droits 700 et la clé rivée en 400
    * on test la connection en mode non bloquant :
`ssh -i ansible -o StrictHostKeyChecking=no ansible@172.17.0.1 'exit 0'`

* le job pour tester ansible:

```
deploy:
  image: ansible/ansible:default
  stage: deploy
  before_script:
    - echo "$ANSIBLE_PKEY" > ansible
    - chmod 700 .
    - chmod 400 ansible
    - pip3 install ansible
  script:
    # - ssh -i ansible -o StrictHostKeyChecking=no ansible@172.17.0.1 'exit 0'
    - ansible -m ping staging
```

* l'installation d'ansible rend le job trop long: on va écrire notre propre image
  * nano Dockerfile dans le dossier utilisateur vagrant pour générer une image custom

````
FROM ansible/ansible:default

RUN pip3 install ansible

ENTRYPOINT ["/usr/bin/env"]

CMD ["/bin/bash"]
````

* on construit la nouvelle image:
    * `docker build -t gitlab.myusine.fr:5050/root/myusine/ansible .`

* on se connecte au registre docker de gitlab avec le compte root
    `docker login gitlab.myusine.fr:5050 (login: root, mdp: roottoor)`

* on la transfère sur le registre de conteneurs (Packages & Registries)
    * `docker push gitlab.myusine.fr:5050/root/myusine/ansible`

* on modifie le job

```
deploy:
  image: gitlab.myusine.fr:5050/root/myusine/ansible
  # image: ansible/ansible:default
  stage: deploy
  before_script:
    - echo "$ANSIBLE_PKEY" > ansible
    - chmod 700 .
    - chmod 400 ansible
#    - pip3 install ansible
  script:
    # - ssh -i ansible -o StrictHostKeyChecking=no ansible@172.17.0.1 'exit 0'
    - ansible -m ping staging
```