

# Premier Pipeline

gitlab / jenkins / docker / ansible / jmeter

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

15- Gitlab trigger

16- Améliorations pour la suite...

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

15- Gitlab trigger

16- Améliorations pour la suite...

# Objectifs ?

- combiner des sujets déjà abordés (cf les playlists)
- quelques définitions devops
- faire un premier squelette d'un pipeline
- préparer l'infrastructure pour supporter notre pipeline
- découvrir de nombreux outils
- échanger sur nos pratiques devops
- deuxième volet : bilan et évolutions / améliorations (factoriser...)

# Objectifs ?

## Infrastructure

- jenkins : jenkins / docker / docker-compose / ansible
- serveur bdd : postgresql
- serveur dev : debian 10 / accès ssh
- serveur stage : debian 10 / accès ssh
- serveur prod : debian 10 / accès ssh

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

15- Gitlab trigger

16- Améliorations pour la suite...

# Devops... pour qui ? pourquoi ?

## **Devops = aire de l'industrialisation de l'IT**

- parallèle avec l'industrie automobile (chaîne de production)
  - avant et après Ford
- parallèle avec le transport maritime (conteneurs)
  - avant et après le conteneur multimodale

# Devops... pour qui ? pourquoi ?

## **Devops = aire de l'industrialisation de l'IT**

- organisation = méthode de travail (agile & co)
- construction d'applicatifs (peu importe le livrable)
  - tar
  - jar
  - bin
  - image OCS
- docker ou pas
- cloud ou pas
- automatisation, standardisation = chaîne de production

# Devops... pour qui ? pourquoi ?

## Devops : culture ou poste ??

- avis diverges : philosophie (attitude) ou poste (réellement identifié)
- dev ou ops ?
  - automatisation = factorisation => ops
  - connaissance des process de code
- adapté à l'environnement de l'entreprise (culture, pratiques, effectifs...)

# Méthode Agile & Scrum

## Organisation

- langage :
  - product owner (chef projet): finalité du produit (besoin client)
  - scrum master : coach
  - backlog : liste des tâches pour atteindre la finalité
  - sprints : période relativement courte pour réaliser des tâches identifiées pour une finalité (2 semaines en moyenne)
  - stand up : réunions courtes quotidiennes (difficultés/avancement)
- Communication visuelle : post-it au mur, site communautaire...

# Fail Fast

- apprentissage par l'échec : l'échec fait partie du processus d'élaboration
- responsabilise tous les acteurs
- un échec tôt dans le processus à un coût moindre
- développer un savoir faire du traitement des échecs
- on attend pas l'échec, on le provoque avec des tests (parallèle avec le chaos monkey)

# Devops... pour qui ? pourquoi ?

## **Devops : la boîte à outils**

- orchestrateur
- ordonnanceur de tâches
- dépôts en tout genre
- testing
- notions databases
- monitoring
- virtualisation (conteneurs, cloud...)
- protocoles web

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

15- Gitlab trigger

16- Améliorations pour la suite...

# Définitions et Outils

## Pipeline ?

- déploiement d'applications : plusieurs...
- automatisation de toutes les tâches
  - dépôts
  - versionning
  - test
  - installation
- combinaison de plusieurs outils

# Définitions et Outils

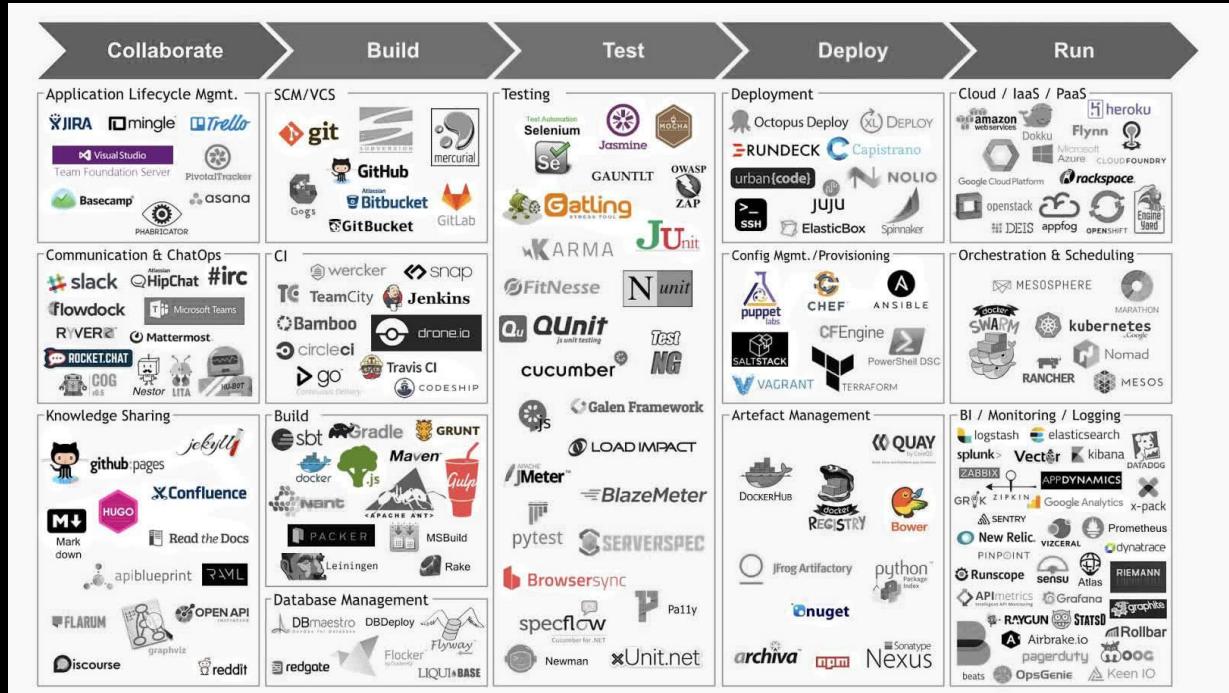
## Pipeline ?



Source : [www.syloë.com](http://www.syloë.com)

# Définitions et Outils

## Pipeline ?



Source : [boris.schapira.dev](http://boris.schapira.dev)

# Définitions et Outils

## Pipeline ?

- **Continuous Integration** : build/test/merge - construction du livrable applicatif
- **Continuous Delivery** : pousse l'applicatif dans son dépôt, dans un état déployable
- **Continuous Deployment** : déploiement (installation) de l'applicatif jusqu'en production

# Définitions et Outils

## Outils ?

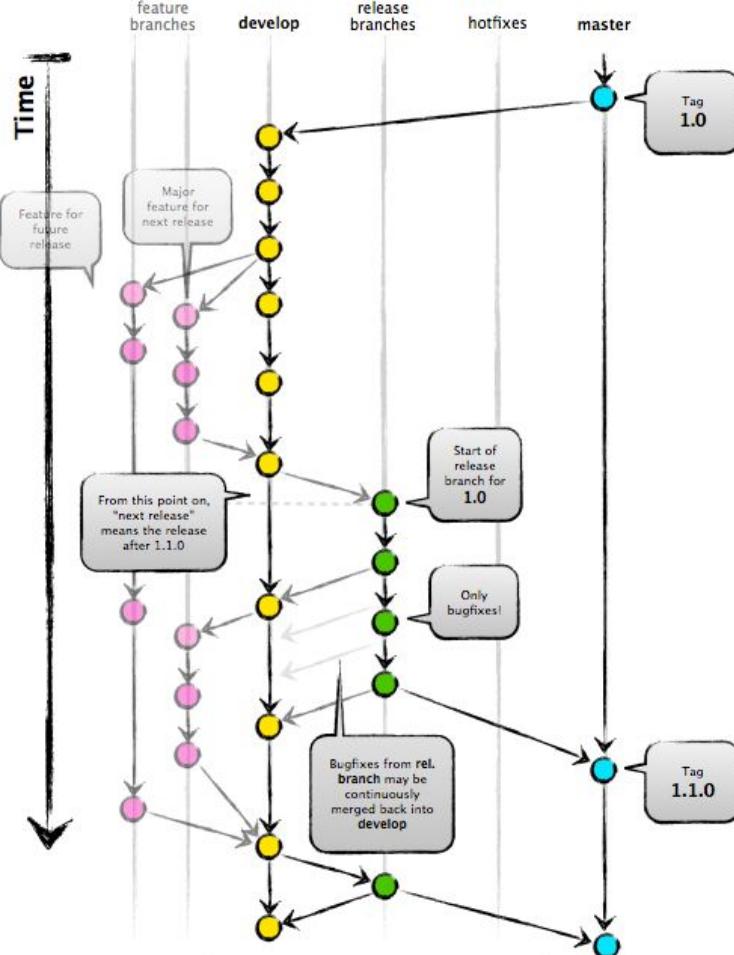
### Technologies variables et évolutives

- scheduler : jenkins
- orchestrateur : ansible
- dépôts : gitlab
- livrable : image docker
- gestionnaire projet : maven

# Définitions et Outils

## Git flow ?

- organisation de développement
- multibranches :  
features/dev/release/master
- évolution par des merges
- existe des variantes
- adaptation au niveau des environnements (serveurs...)



# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

15- Gitlab trigger

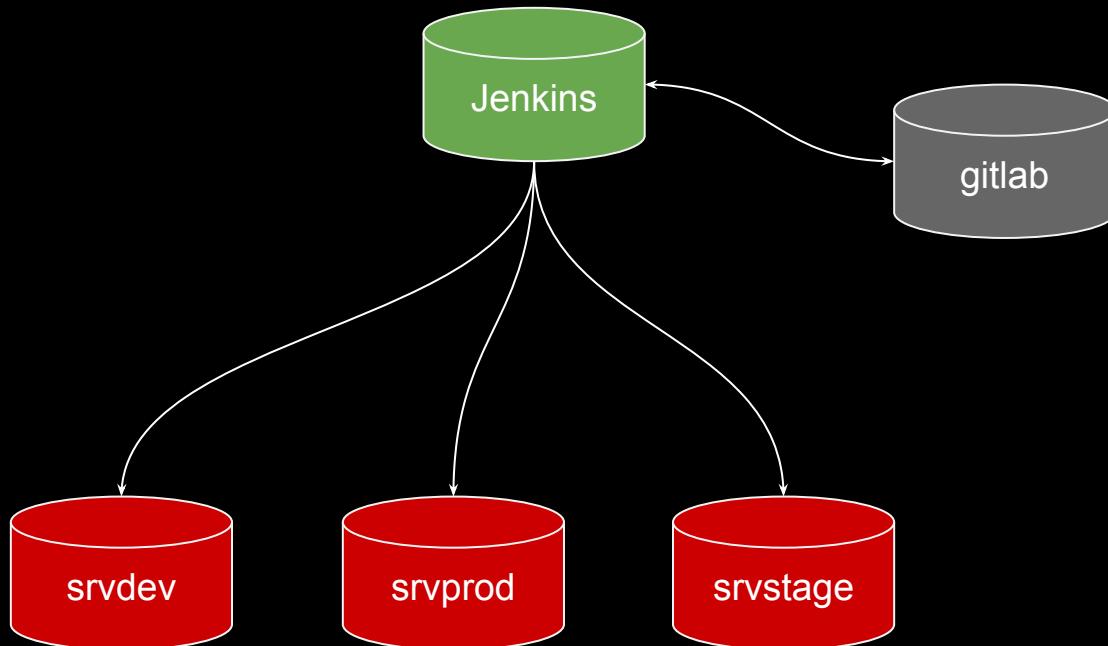
16- Améliorations pour la suite...

# Infrastructure

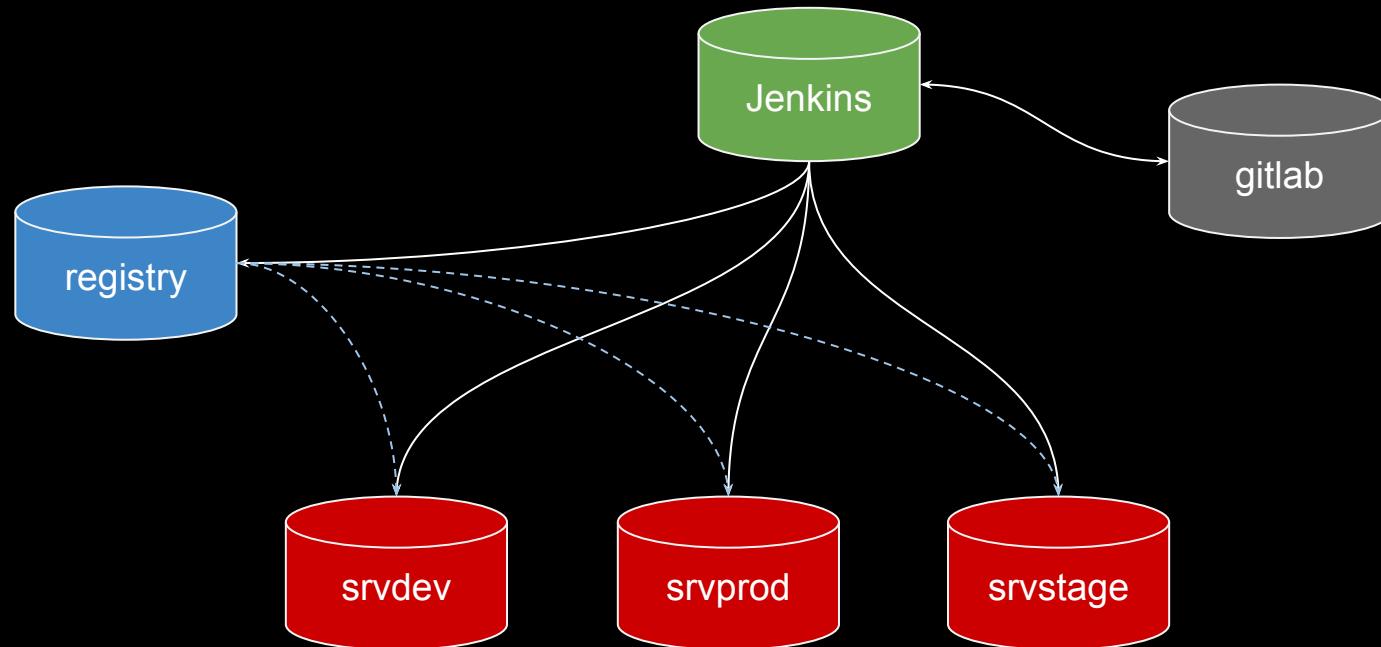
## Infrastructure

- jenkins : jenkins / docker / docker-compose / ansible
- gitlab
- registry docker
- serveur bdd : postgresql
- serveur dev : debian 10 / accès ssh
- serveur stage : debian 10 / accès ssh
- serveur prod : debian 10 / accès ssh

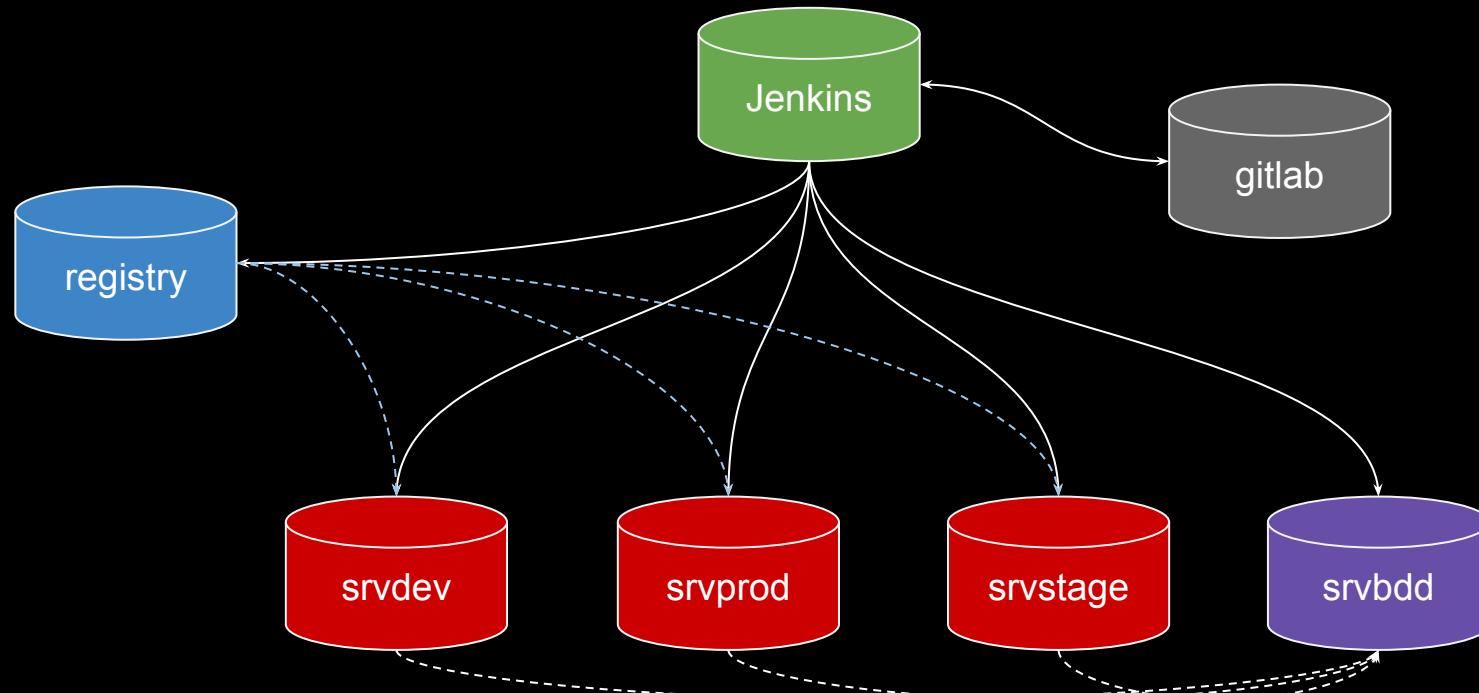
# Infrastructure



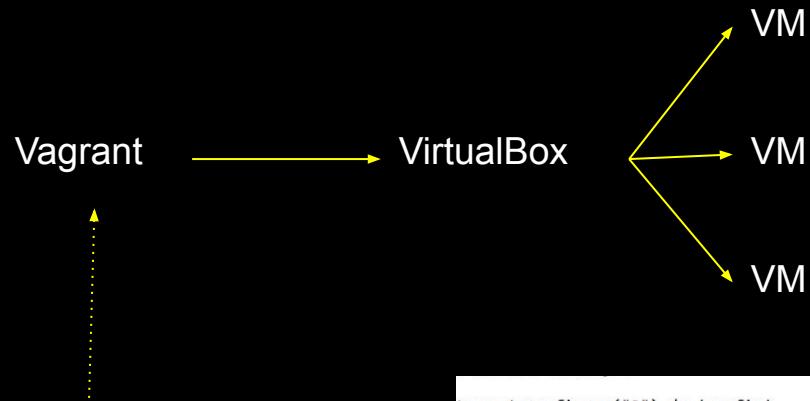
# Infrastructure



# Infrastructure



# Infrastructure



- Fichiers descriptifs (ruby)
- Scripts (ou ansible)

```
/agrant.configure("2") do |config|
  # jenkins server
  config.vm.define "jenkins-pipeline" do |jenkins|
    jenkins.vm.box = "debian/buster64"
    jenkins.vm.hostname = "jenkins-pipeline"
    jenkins.vm.box_url = "debian/buster64"
    jenkins.vm.network :private_network, ip: "192.168.10.2"
    jenkins.vm.provider :virtualbox do |v|
      v.customize ["--modifyvm", :id, "--natdnshostresolver1", "on"]
      v.customize ["--modifyvm", :id, "--natdnspolicy1", "on"]
      v.customize ["--modifyvm", :id, "--memory", 3072]
      v.customize ["--modifyvm", :id, "--name", "jenkins-pipeline"]
      v.customize ["--modifyvm", :id, "--cpus", "2"]
    end
    config.vm.provision "shell", inline: <<-SHELL
      sed -i 's/challengeResponseAuthentication yes/g' /etc/ssh/sshd_config
      service ssh restart
    SHELL
    jenkins.vm.provision "shell", path: "install_jenkins.sh"
  end
```

# Infrastructure

## Vagrant : commandes

<https://www.vagrantup.com/>

- vagrant up : download image & run
- vagrant up <serveur> : lancer un serveur seul
- vagrant snapshot push : faire des snapshots
- vagrant snapshot list
- vagrant snapshot pop : charger un snapshot
- vagrant halt : arrêter la ou les machines
- vagrant destroy : détruire la ou les machines (-f)
- vagrant status : liste des machines
- cat Vagrantfile | grep -ri "ip:" : liste des ips

# Infrastructure

## Vagrant : installation

- Installation de virtualbox

*sudo apt install virtualbox*

- Installation de vagrant par apt

*sudo apt install vagrant*

- ou installation de vagrant par dpkg

<https://www.vagrantup.com/downloads.html>

*sudo wget https://releases.hashicorp.com/vagrant/2.2.6/vagrant\_2.2.6\_x86\_64.deb*

*sudo dpkg -i vagrant\_2.2.6\_x86\_64.deb*

# Infrastructure

## Vagrant : installation Jenkins

- HW : 2 cpus / 3072 M ram
- java : default-jre (attention compatibilité de version si maven hors docker)
- jenkins (sans docker sinon docker in docker...)
- ansible / sshpass / gpg
- git
- docker : pour builder et pour préparer les environnements
- tricks
  - ansible : pipelining true et allow readable tmp true
  - docker : usermod -aG docker jenkins
  - registry : insecure registries

# Infrastructure

## Vagrant : Jenkins

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  # jenkins server
  config.vm.define "jenkins-pipeline" do |jenkins|
    jenkins.vm.box = "debian/buster64"
    jenkins.vm.hostname = "jenkins-pipeline"
    jenkins.vm.box_url = "debian/buster64"
    jenkins.vm.network :private_network, ip: "192.168.10.2"
    jenkins.vm.provider :virtualbox do |v|
      v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
      v.customize ["modifyvm", :id, "--natdnspoolproxy1", "on"]
      v.customize ["modifyvm", :id, "--memory", 3072]
      v.customize ["modifyvm", :id, "--name", "jenkins-pipeline"]
      v.customize ["modifyvm", :id, "--cpus", "2"]
    end
    config.vm.provision "shell", inline: <<-SHELL
      sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd_config
      service ssh restart
    SHELL
    jenkins.vm.provision "shell", path: "install_jenkins.sh"
  end
end
```

# Infrastructure

## Vagrant : installation Jenkins

```
IP=$(hostname -I | awk '{print $2}')

echo "START - install jenkins - \"$IP"
Fichier Edition Afficher Insertion Format Diapositive Réorganiser Outils Modules complémentaires Aide Trouver les modifications ont été enregistrées dans Drive
echo "[1]: install jenkins"
apt-get update -qq >/dev/null
apt-get install -qq -y git sshpass wget ansible gnupg2 curl git >/dev/null
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
apt-get update -qq >/dev/null
apt-get install -qq -y default-jre jenkins >/dev/null
systemctl enable jenkins
systemctl start jenkins

sed -i 's/.*pipelining.*/pipelining = True/' /etc/ansible/ansible.cfg
sed -i 's/.*allow_world_readable_tmpfiles.*/allow_world_readable_tmpfiles = True/' /etc/ansible/ansible.cfg

echo "START - install docker - \"$IP"
curl -fsSL https://get.docker.com | sh; >/dev/null
usermod -aG docker jenkins
curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
echo "
{
  \"insecure-registries\" : [\"192.168.10.5:5000\"]
}
" >/etc/docker/daemon.json
systemctl daemon-reload
systemctl restart docker

echo "END - install jenkins"
```

# Infrastructure

## Vagrant : serveurs applicatif x 3

- 3 environnements : dev / stage / prod(master)
- HW : 1 cpu / 512M ram
- debian buster
- sans docker
- sans java
- connexions ssh user : vagrant / mdp : vagrant

# Infrastructure

## Vagrant : serveurs applicatif x 3 (dev/stage/prod)

```
config.vm.define "srvdev" do |srvdev|
  srvdev.vm.box = "debian/buster64"
  srvdev.vm.hostname = "srvdev"
  srvdev.vm.box_url = "debian/buster64"
  srvdev.vm.network :private_network, ip: "192.168.10.3"
  srvdev.vm.provider :virtualbox do |v|
    v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
    v.customize ["modifyvm", :id, "--natdnsproxy1", "on"]
    v.customize ["modifyvm", :id, "--memory", 512]
    v.customize ["modifyvm", :id, "--name", "srvdev"]
    v.customize ["modifyvm", :id, "--cpus", "1"]
  end
  config.vm.provision "shell", inline: <<-SHELL
    sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd_config
    service ssh restart
  SHELL
end
```

# Infrastructure

## Vagrant : serveur bdd (postgres)

- 1 serveur unique pour 3 env / gestion par bdd (pas représentatif)
- HW : 1 cpu / 512 M ram
- postgresql
- user : vagrant / mdp : vagrant (1 seul pas top)
- 3 bases de données : dev / stage / prod
- /etc/.../postgresql.conf : listen all “\*”
- /etc/.../pg\_hba.conf : open for all 127.0.0.1/32 => 0.0.0.0/0 (à ne pas faire)

# Infrastructure

## Vagrant : serveur bdd (postgres)

```
config.vm.define "srvbdd" do |srvbdd|
  srvbdd.vm.box = "debian/buster64"
  srvbdd.vm.hostname = "srvbdd"
  srvbdd.vm.box_url = "debian/buster64"
  srvbdd.vm.network :private_network, ip: "192.168.10.6"
  srvbdd.vm.provider :virtualbox do |v|
    v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
    v.customize ["modifyvm", :id, "--natdnsp proxy1", "on"]
    v.customize ["modifyvm", :id, "--memory", 512]
    v.customize ["modifyvm", :id, "--name", "srvbdd"]
    v.customize ["modifyvm", :id, "--cpus", "1"]
  end
  config.vm.provision "shell", inline: <<-SHELL
    sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd_config
    service ssh restart
  SHELL
  srvbdd.vm.provision "shell", path: "install_srvbdd.sh"
end
```

# Infrastructure

## Vagrant : installation serveur bdd (postgres)

```
#!/bin/bash
## install master consul ab

IP=$(hostname -I | awk '{print $2}')
echo "START - install postgres - \"$IP\""
echo "[1]: install postgres"
apt-get update -qq >/dev/null
apt-get install -qq -y vim git curl git >/dev/null
apt-get install -qq postgresql-11 >/dev/null
sudo -u postgres bash -c "psql -c \"CREATE USER vagrant WITH PASSWORD 'vagrant';\""
sudo -u postgres bash -c "psql -c \"CREATE DATABASE dev OWNER vagrant;\""
sudo -u postgres bash -c "psql -c \"CREATE DATABASE stage OWNER vagrant;\""
sudo -u postgres bash -c "psql -c \"CREATE DATABASE prod OWNER vagrant;\""
sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/" /etc/postgresql/11/main/postgresql.conf
sed -i "s/127.0.0.1\|32/0.0.0.0\|0/g" /etc/postgresql/11/main/pg_hba.conf
service postgresql restart

echo "END - install postgres"
```

# Infrastructure

## Vagrant : registry

- HW : 1 cpu / 512 M ram
- docker
- docker-compose
- certificat ssl
- génération password (docker exec)
- édition de docker-compose.yml
- docker-compose up

# Infrastructure

## Vagrant : registry

```
config.vm.define "registry" do |registry|  
  registry.vm.box = "debian/buster64"  
  registry.vm.hostname = "registry"  
  registry.vm.box_url = "debian/buster64"  
  registry.vm.network :private_network, ip: "192.168.10.5"  
  registry.vm.provider :virtualbox do |v|  
    v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]  
    v.customize ["modifyvm", :id, "--natdnsproxy1", "on"]  
    v.customize ["modifyvm", :id, "--memory", 512]  
    v.customize ["modifyvm", :id, "--name", "registry"]  
    v.customize ["modifyvm", :id, "--cpus", "1"]  
  end  
  config.vm.provision "shell", inline: <<-SHELL  
    sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd_config  
    service ssh restart  
  SHELL  
  registry.vm.provision "shell", path: "install_registry.sh"  
end
```

# Infrastructure

## Vagrant : installation registry

```
apt-get update -qq >/dev/null
apt-get install -qq -y git wget curl git >/dev/null
curl -fsSL https://get.docker.com | sh; >/dev/null
curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
Reorganiser Outils Modules complémentaires Aide Toutes les modifications ont été acceptées dans l'ordre
echo "[2]: install registry"
mkdir certs/
openssl req -x509 -nodes -newkey rsa:4096 -keyout certs/myregistry.key -out certs/myregistry.crt -days 365 -subj /CN=myregistry.my
mkdir passwd/
docker run --entrypoint htpasswd registry:2 -Bbn xavki password > passwd/htpasswd

mkdir data/
echo "
version: '3.5'
services:
  registry:
    restart: always
    image: registry:2
    container_name: registry
    ports:
      - 80:5000
    environment:
      REGISTRY_HTTP_TLS_CERTIFICATE: /certs/myregistry.crt
      REGISTRY_HTTP_TLS_KEY: /certs/myregistry.key
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
    volumes:
      - ./data:/var/lib/registry
      - ./certs:/certs
      - ./passwd:/auth
" > docker-compose-registry.yml
docker-compose -f docker-compose-registry.yml up -d
```

# Infrastructure

## Vagrant : gitlab

- HW : 1 cpu / 2048 M ram
- installation package (choix avec docker)
- peut-être configuration des locales

# Infrastructure

## Vagrant : gitlab

```
config.vm.define "gitlab" do |gitlab|
  gitlab.vm.box = "debian/buster64"
  gitlab.vm.hostname = "gitlab"
  gitlab.vm.box_url = "debian/buster64"
  gitlab.vm.network :private_network, ip: "192.168.10.10"
  gitlab.vm.provider :virtualbox do |v|
    v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
    v.customize ["modifyvm", :id, "--natdnsproxy1", "on"]
    v.customize ["modifyvm", :id, "--memory", 2048]
    v.customize ["modifyvm", :id, "--name", "gitlab"]
    v.customize ["modifyvm", :id, "--cpus", "1"]
  end
  config.vm.provision "shell", inline: <<-SHELL
    sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd_config
    service ssh restart
  SHELL
  gitlab.vm.provision "shell", path: "install_gitlab.sh"
end
end
```

# Infrastructure

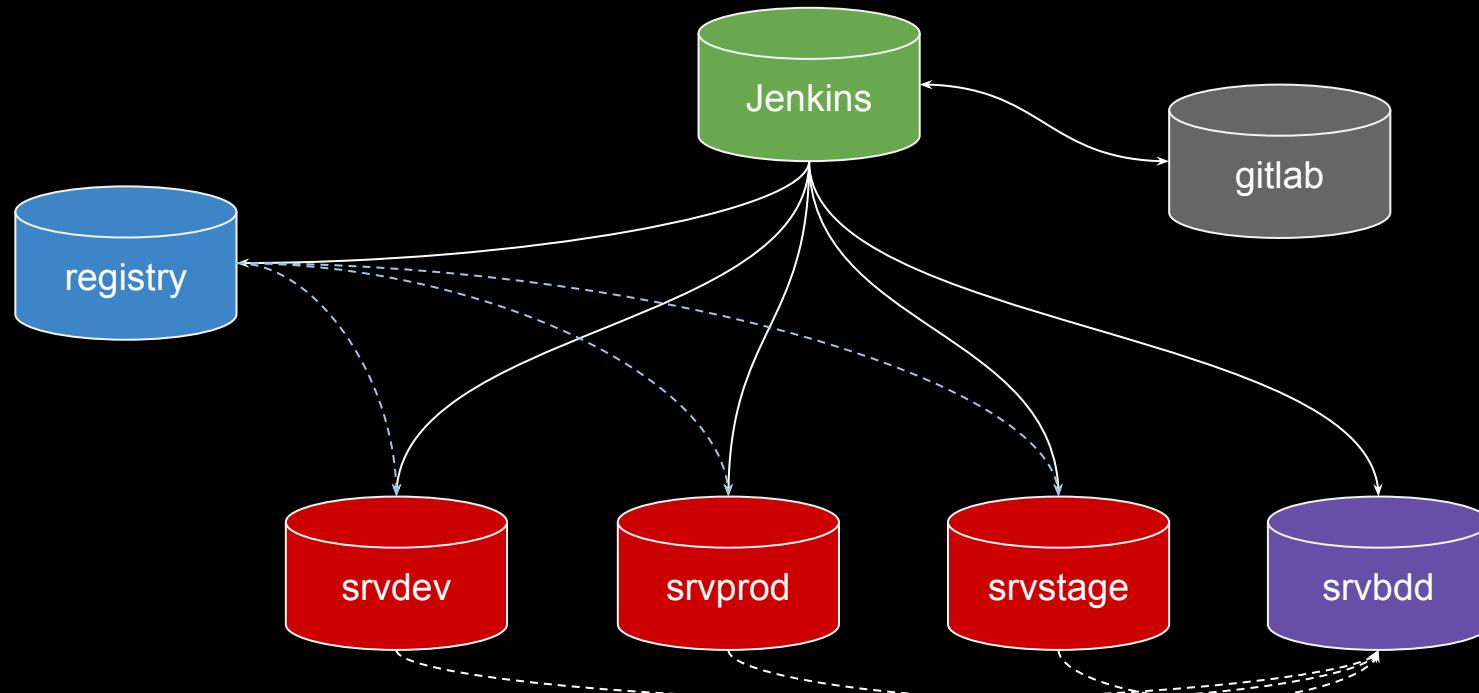
## Vagrant : installation gitlab

```
#!/bin/bash
IP=$(hostname -I | awk '{print $2}')
echo "START - install gitlab - \"$IP\""
echo "[1]: install gitlab"
apt-get update -qq >/dev/null
apt-get install -qq -y vim wget curl git >/dev/null
curl -sS https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash
apt-get update -qq >/dev/null
export LC_CTYPE=en_US.UTF-8
export LC_ALL=en_US.UTF-8
apt install -y gitlab-ce
echo "END - install gitlab"

#sudo dpkg-reconfigure locales
```

Vagrant : installation gitlab

# Infrastructure



# Infrastructure

## Quelques mots : demo vs prod

- réseau : isolation (vlan, ouvertures de flux)
- jenkins : ansible accès sudo, isolation (1 par environnement), slaves
- base de données :
  - un serveur par environnement
  - sécuriser le pg\_hba et le listen
  - gestion des droits sql et users
  - backup/réPLICATION
- registry/gitlab : isolation production ?
- application : reverse-proxy devant, load-balancing, mesh...

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

15- Gitlab trigger

16- Améliorations pour la suite...

# Applicatif

## Le choix ?

- représentatif d'un cas d'utilisation sans complexité
- réunir un cas d'utilisation assez complet (aller au-delà du simple build d'image docker et test unitaire) => langage compilé
- pouvoir tester facilement pour utiliser des outils de tests => API
- faire appel à une base de données => PostgreSQL

# Applicatif

## Java => Spring Boot + Hibernate + JPA

- **Sprint Boot :**
  - framework facilitant le développement d'applications java (notamment web)
  - permet le packaging en jar
- **Hibernate :**
  - framework facilitant la gestion d'objets de type relationnel
  - facilite l'utilisation de objets base de données par des méthodes
- **Java Persistence API :**
  - facilite le passage d'objets relationnels sous forme d'API
  - permet d'ajouter une couche d'abstraction avec les annotations Java

# Applicatif

## Fichiers/Répertoires

- pom.xml
- dir src/main/ et dir src/test/
- fichier propriétés :
  - src/main/resources/application.properties
  - connexion bdd
  - port...
- class principale : src/main/java/.../PostgresDemoApplication.java

# Applicatif

## Fichier application.properties

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url=jdbc:postgresql://192.168.168.2:5432/mydb
spring.datasource.username= myuser
spring.datasource.password= myuserpassword

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

15- Gitlab trigger

16- Améliorations pour la suite...

# Pipeline : ébauche

## Build applicatif

Environnement

Test unitaire



**Git clone generator**

Git clone

Script de génération

d'environnement :

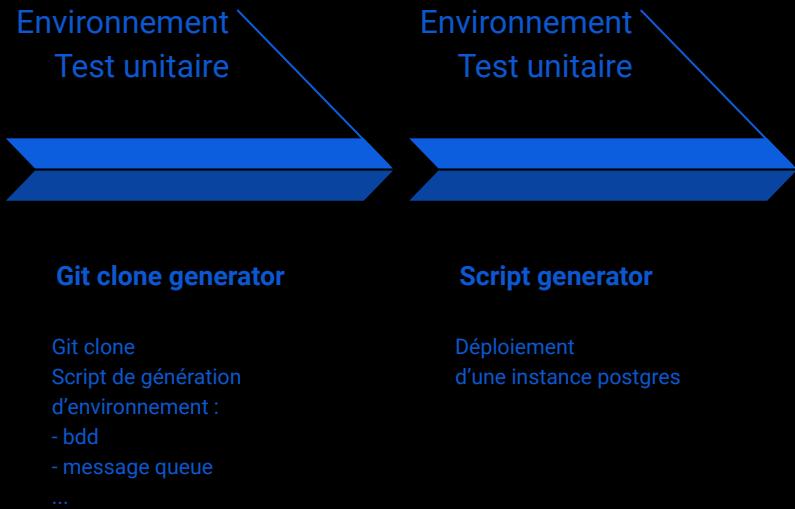
- bdd

- message queue

...

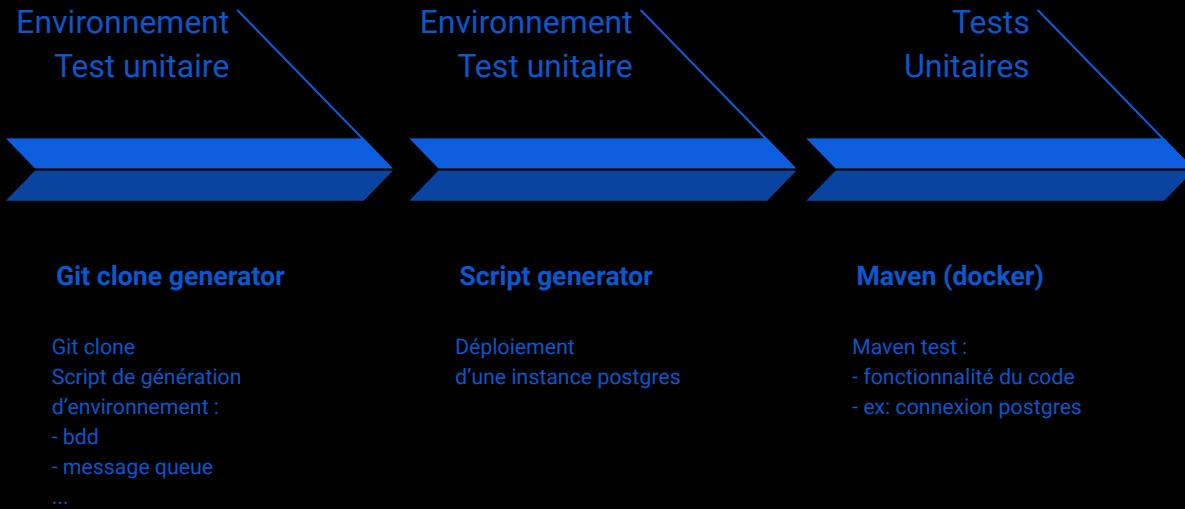
# Pipeline : ébauche

## Build applicatif



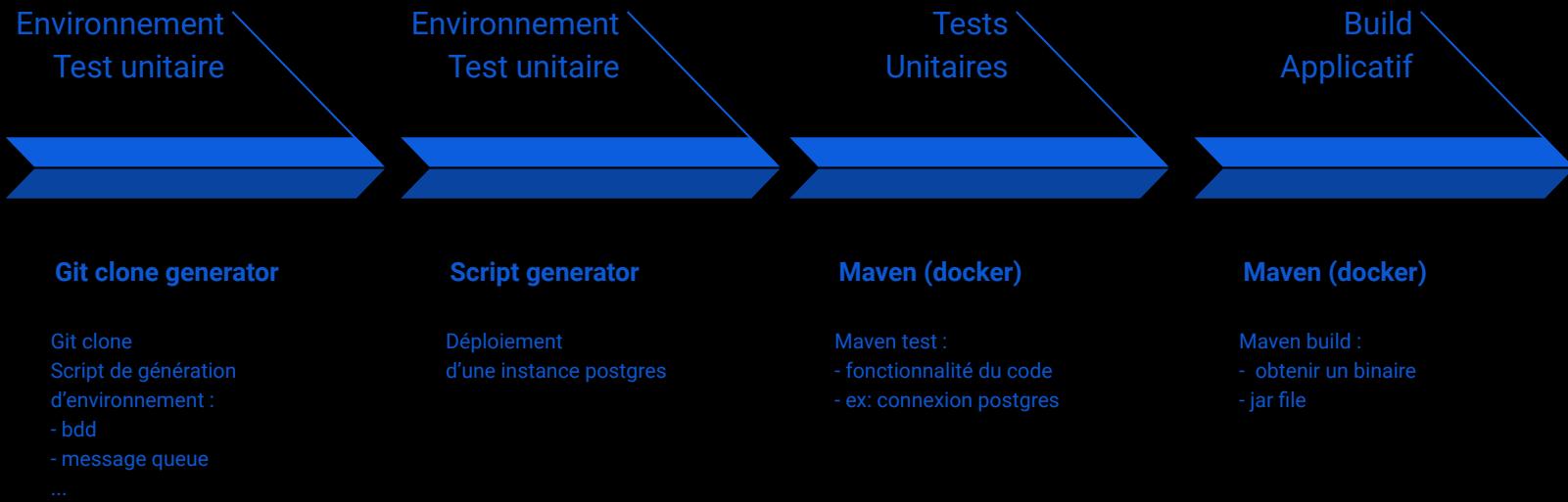
# Pipeline : ébauche

## Build applicatif



# Pipeline : ébauche

## Build applicatif



# Pipeline : ébauche

## Build Docker



**Docker build**

Chargement du JAR dans une  
image openjdk

# Pipeline : ébauche

## Build Docker



### Docker build

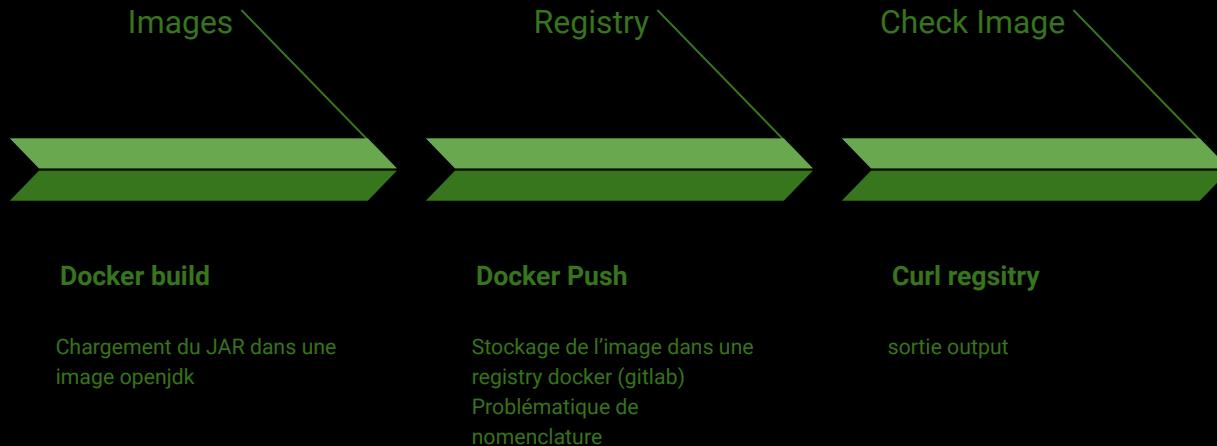
Chargement du JAR dans une image openjdk

### Docker Push

Stockage de l'image dans une registry docker (gitlab)  
Problématique de nomenclature

# Pipeline : ébauche

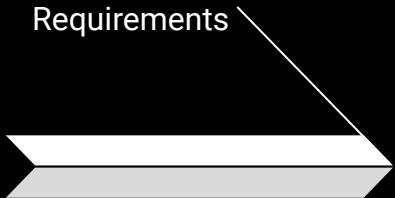
## Build Docker



# Pipeline : ébauche

## Ansible

Requirements

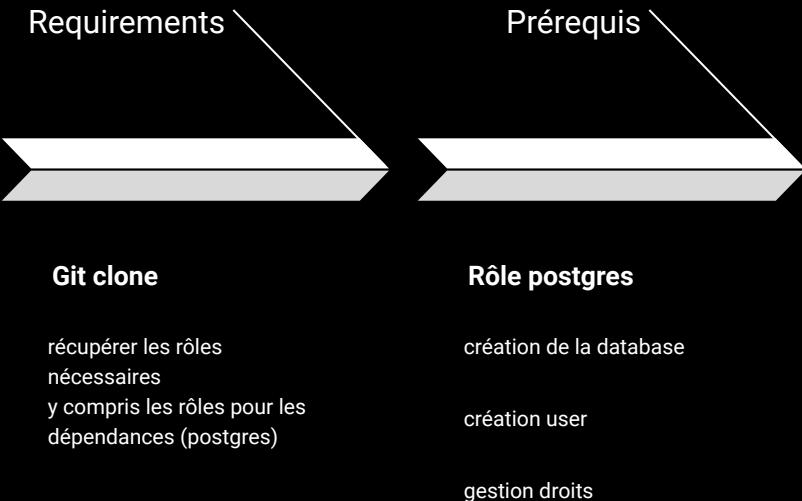


### Git clone

récupérer les rôles  
nécessaires  
y compris les rôles pour les  
dépendances (postgres)

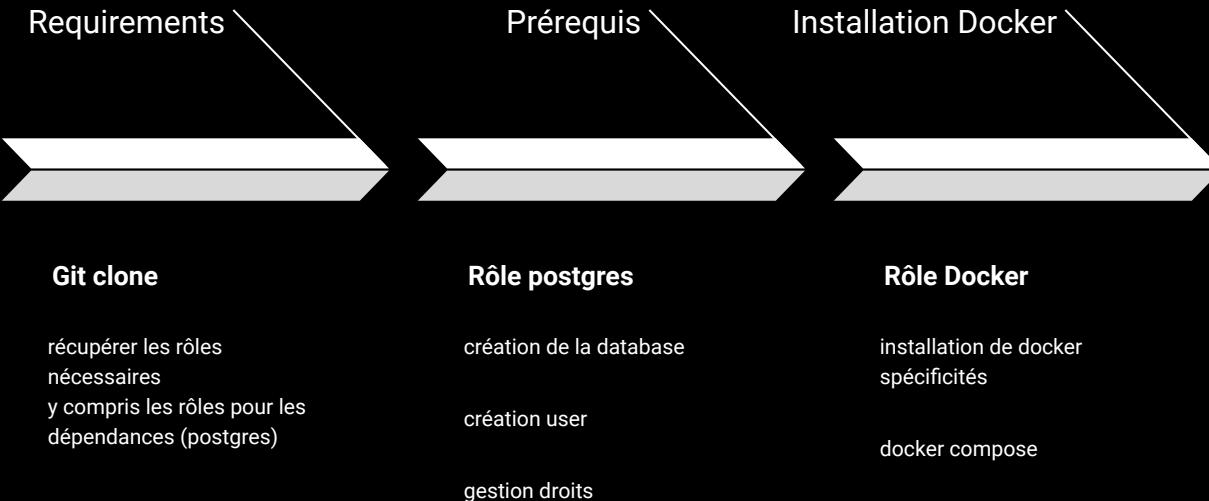
# Pipeline : ébauche

## Ansible



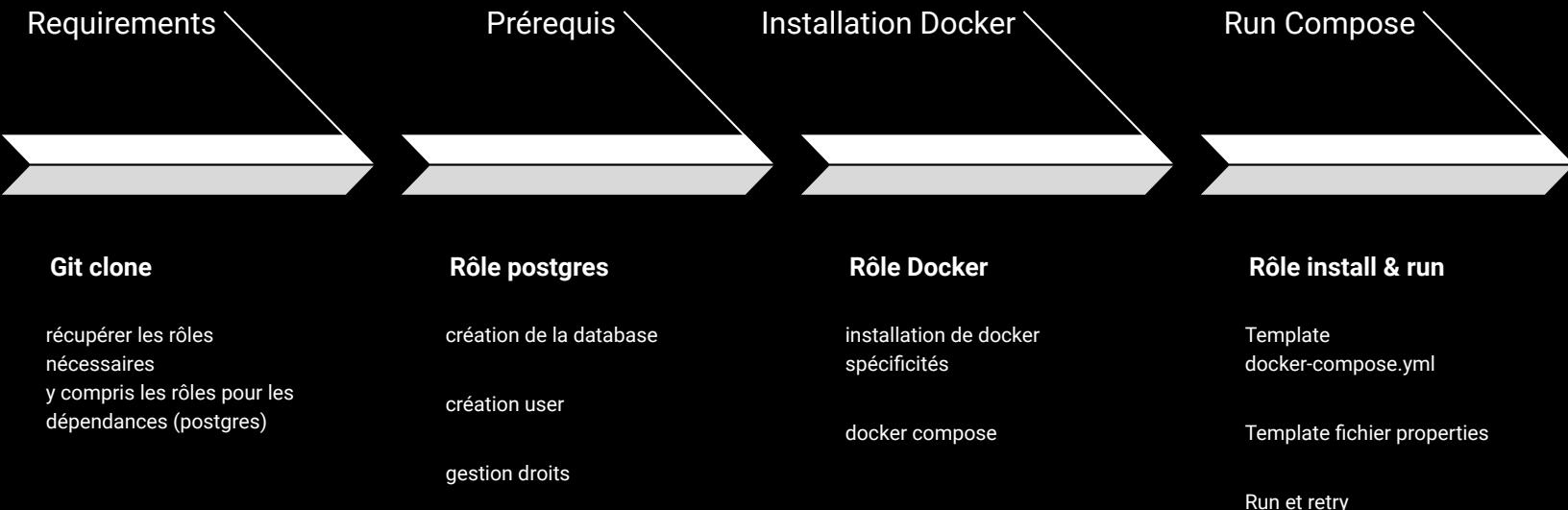
# Pipeline : ébauche

## Ansible



# Pipeline : ébauche

## Ansible



# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Script environnement : bdd

10- Maven : principes, tests, build

11- Docker : dockerfile, push

12- Jenkins : jenkinsfile

13- Ansible : déploiement global

14- Jmeter... pour quelques tests

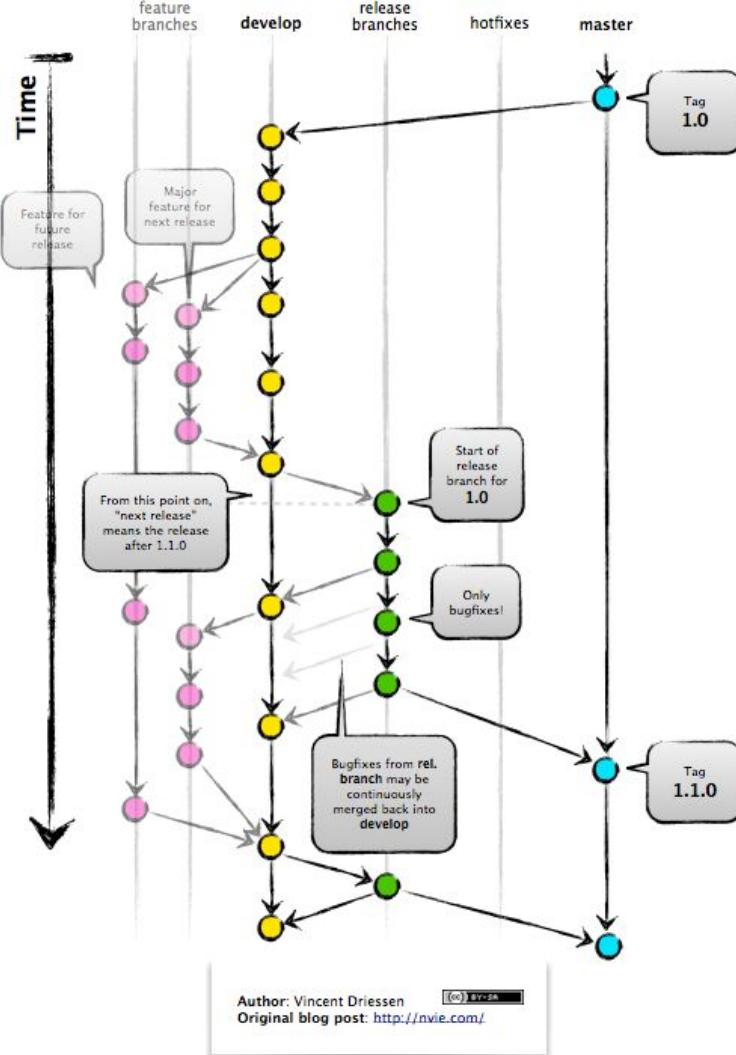
15- Gitlab trigger

16- Améliorations pour la suite...

# Définitions et Outils

## Git flow ?

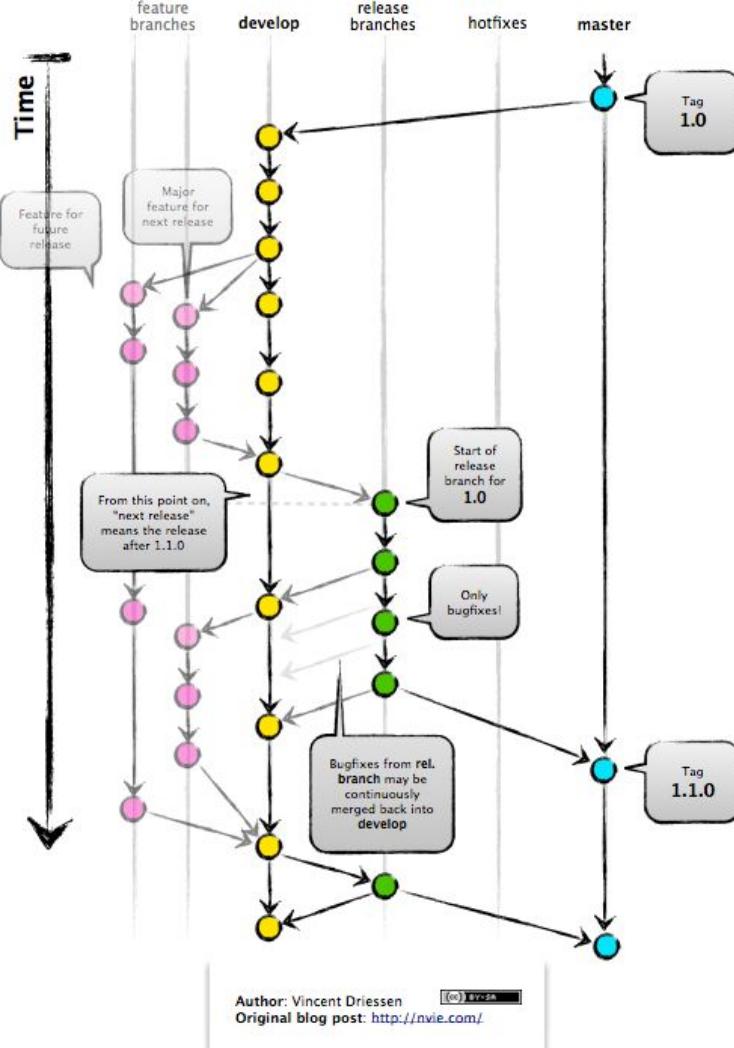
- organisation de développement
- multibranches :  
features/dev/release/master
- évolution par des merges
  - merge request gitlab
- existe des variantes
- adaptation au niveau des environnements (serveurs...)



# Définitions et Outils

## CLI Git

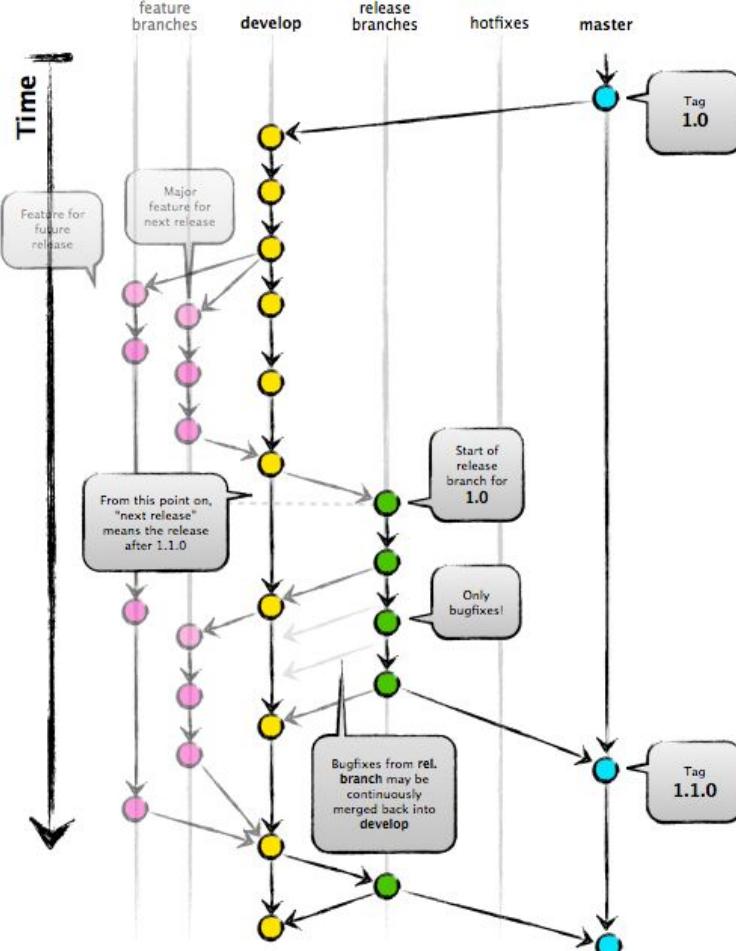
- git clone : télécharger un dépôt
- git init : initialisation d'un dépôt
- git add <fichiers> : ajout de fichier à commiter
- git commit -m "commentaire"
- git commit -amend : modifier un commentaire
- git push : pousser les fichiers sur le remote
- git pull : resynchronisation avec le remote



# Définitions et Outils

## CLI Git

- git branch <nom\_branche>
- git checkout <nom\_branche>
- git remote add origin <url>
- git push -u origin master
- git tag -a v1.4 -m "my version 1.4"
- git push origin <nom\_tag> (ou --tags)



# Définitions et Outils

## GitFlow en pratique

Dev

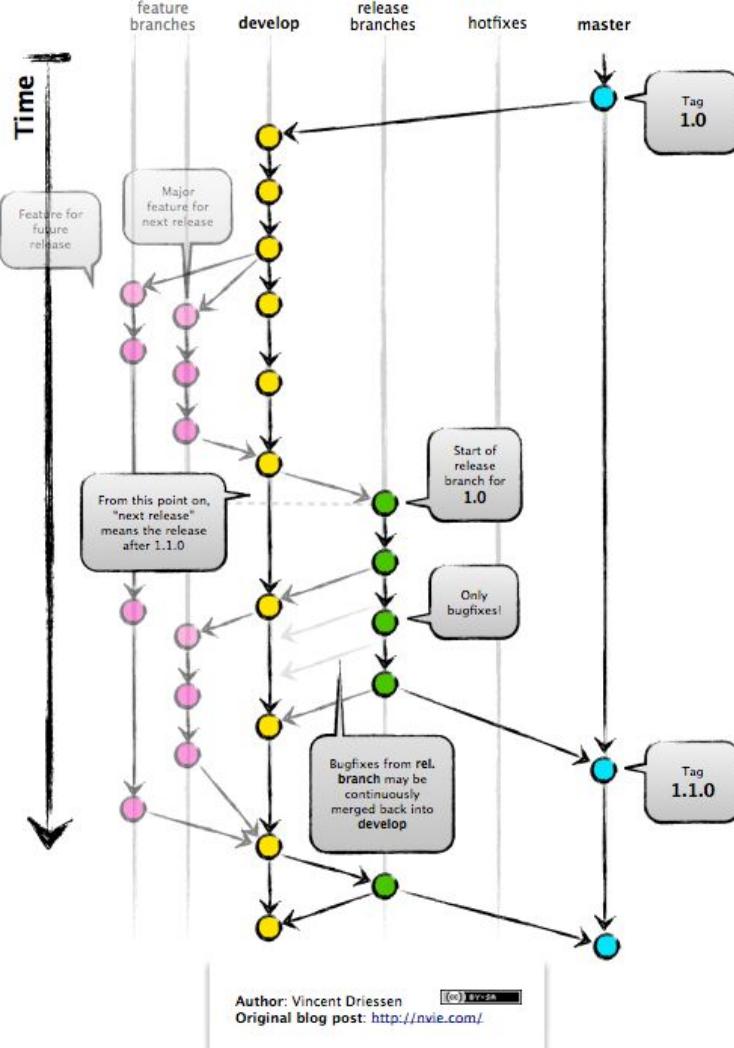
- git pull > modification
- git add > git commit > git push  
(... merge request)

Recette

- git checkout > git merge dev (sauf si MR)
- git push > tests > (... merge request) > git tag

Prod

- git merge recette
- git push



# Sommaire

- 1- Objectifs
- 2- Devops : pkoi, agile, failfast...
- 3- Définitions et Outils
- 4- Infrastructure : vagrant...
- 5- Applicatif : choix
- 6- Pipeline : ébauche
- 7- Git : cli, gitflow
- 8- Gitlab : présentation, dépôts

# GitLab

## Commande GitLab-ctl

- gitlab-ctl status
- gitlab-ctl start
- gitlab-ctl stop
- gitlab-ctl restart
- gitlab-ctl restart nginx
- gitlab-psql -d gitlabhq\_production
- gitlab-rails console

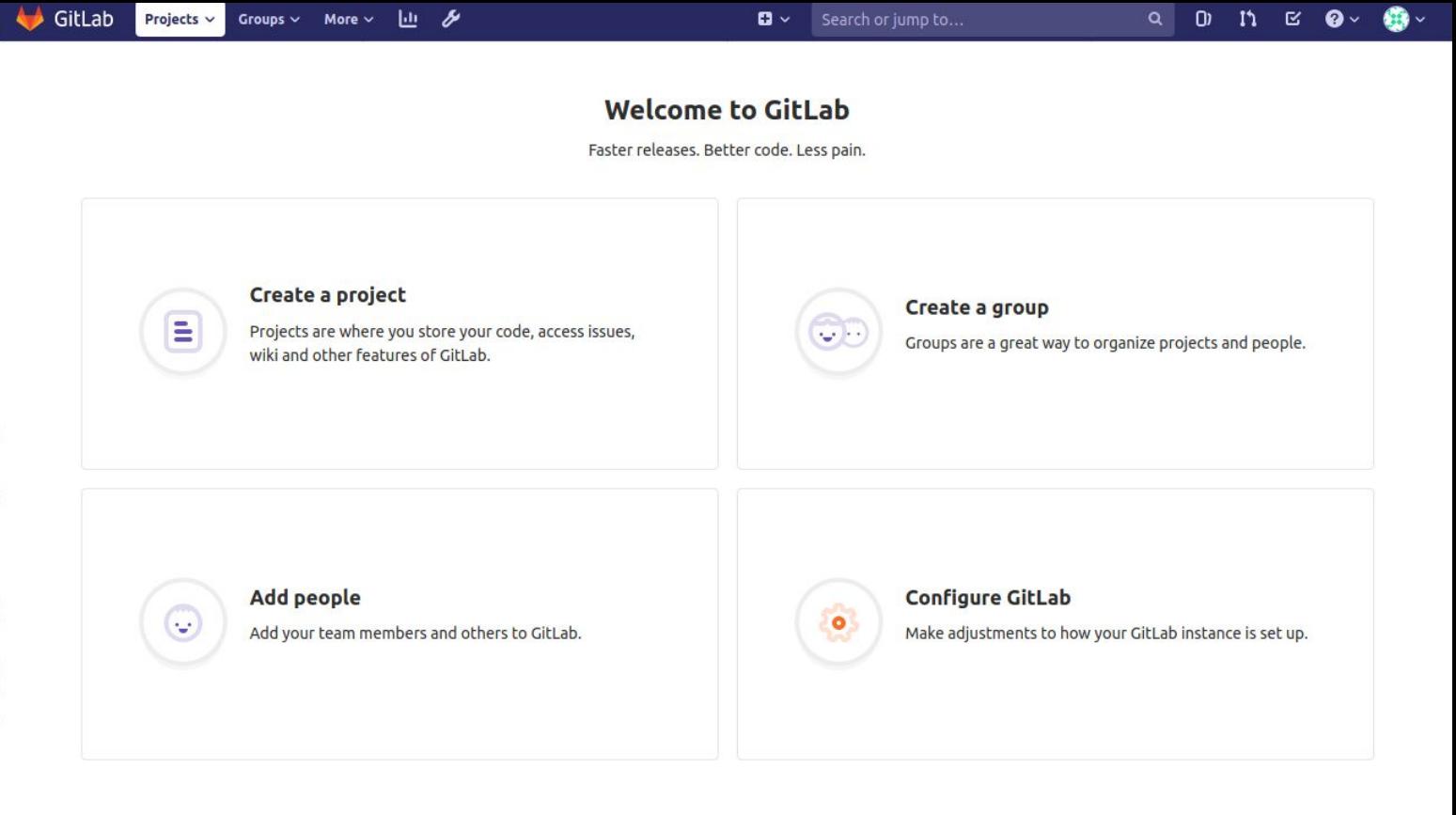
# GitLab

## Un outil plus que polyvalent

- outil simple en apparence... complexe sous le capot : rails, nginx, redis, prometheus, grafana, postgresql, node exporter...
- organisation : issues, tasks, dashboard
- documentation : wiki et pages
- dépôt git : commit, statistiques, merge request (liens avec les issues...)
- docker registry : gestion des droits, classement par projet (tris et clean délicats)
- GitLab-CI : concurrent à Jenkins (intégration grafana, runner, autodevops...)

=> uniquement dépôt pour gérer notre gitflow

# GitLab



The image shows the GitLab welcome screen. At the top, there is a dark blue header bar with the GitLab logo, a search bar containing "Search or jump to...", and various navigation icons. Below the header, the main content area has a white background with a central title and four call-to-action boxes.

**Welcome to GitLab**

Faster releases. Better code. Less pain.

**Create a project**  
Projects are where you store your code, access issues, wiki and other features of GitLab.

**Create a group**  
Groups are a great way to organize projects and people.

**Add people**  
Add your team members and others to GitLab.

**Configure GitLab**  
Make adjustments to how your GitLab instance is set up.

## Users & Projets

- création d'un user
- ajout de la clef ssh
- création d'un projet
- ajout d'un user à un projet / permission

# GitLab

## Dépôt Git

- création/reprise d'un dépôt
- merge request
- statistiques
- autres outils : wiki, issues...

## Démo : plusieurs dépôts

- générateur d'environnement : script generator de compose (pour postgres en test unitaires)
- applicatif (usage de gitflow) : code applicatif + Jenkinsfile + Dockerfile
- ansible : playbook de déploiement et variables
- un dépôt par rôle ansible (gitflow) :
  - postgres
  - docker / docker-compose
  - applicatif

# Sommaire

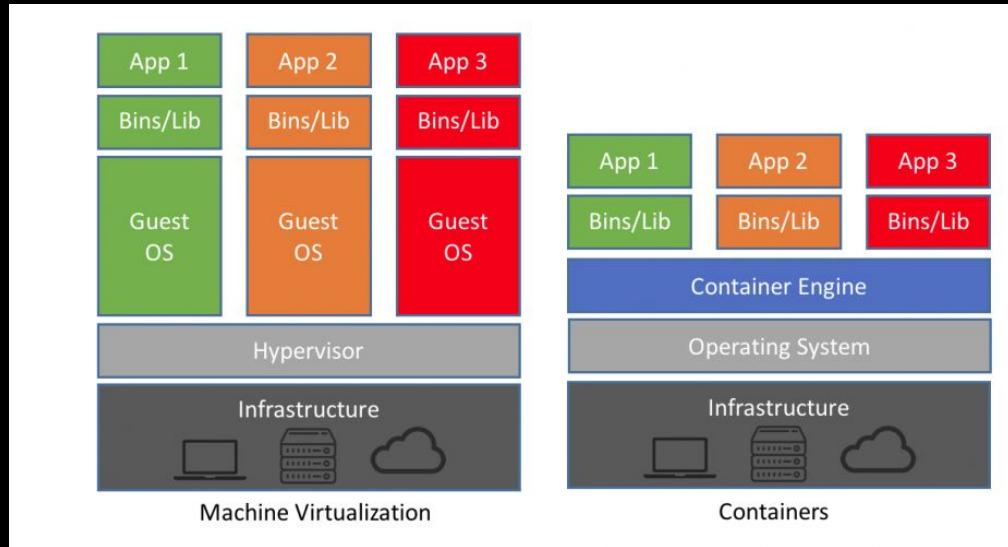
## 9- Docker rappels

- 1- Objectifs
- 2- Devops : pkoi, agile, failfast...
- 3- Définitions et Outils
- 4- Infrastructure : vagrant...
- 5- Applicatif : choix
- 6- Pipeline : ébauche
- 7- Git : cli, gitflow
- 8- Gitlab : présentation, dépôts

# Rappel sur les conteneurs et Docker

## Conteneurs vs VM

- plus léger



# Rappel sur les conteneurs et Docker

## Quelques rappels sur docker et les conteneurs

- images : un ensemble de fichiers, non modifiables, “sans vie” (sans process), composée de librairies et binaires et qui va être lancer pour devenir un conteneur (avec un process principal)
- conteneurs : une image runnée, c'est à dire lancement du process principal, 1 image = 1/+ conteneurs, peut avoir des variables, réseaux et volumes spécifiques

# Rappel sur les conteneurs et Docker

## Intérêts des conteneurs & Docker

- livrable simple, versionning
- agnostique : inclure les dépendances
- installation sur différents supports :
  - kubernetes, swarm, nomad, docker host
- dépôt registry avec authentification
- plusieurs utilisations dans le pipeline

# Rappel sur les conteneurs et Docker

## Docker CLI

- configuration moteur : /etc/docker/daemon.json
  - logs, ip, remap user, proxy...
- docker --help
- docker info : information de notre moteur
- docker <commande\_linux> (ps, ls...)
- docker inspect <id\_element> : metadata

# Rappel sur les conteneurs et Docker

## Docker image

- docker pull <image>
  - notion de latest
- docker login <url\_registry>
- docker push <url>/<image>:<tags>
- docker pull <url>/<image>:<tags> (défaut)
- docker image ls (images)
- docker image inspect <image>

# Rappel sur les conteneurs et Docker

## Docker image

- docker pull <image>
  - notion de latest
- docker login <url\_registry>
- docker push <url>/<image>:<tags>
- docker pull <url>/<image>:<tags> (défaut)
- docker image ls (images)
- docker image inspect <image>

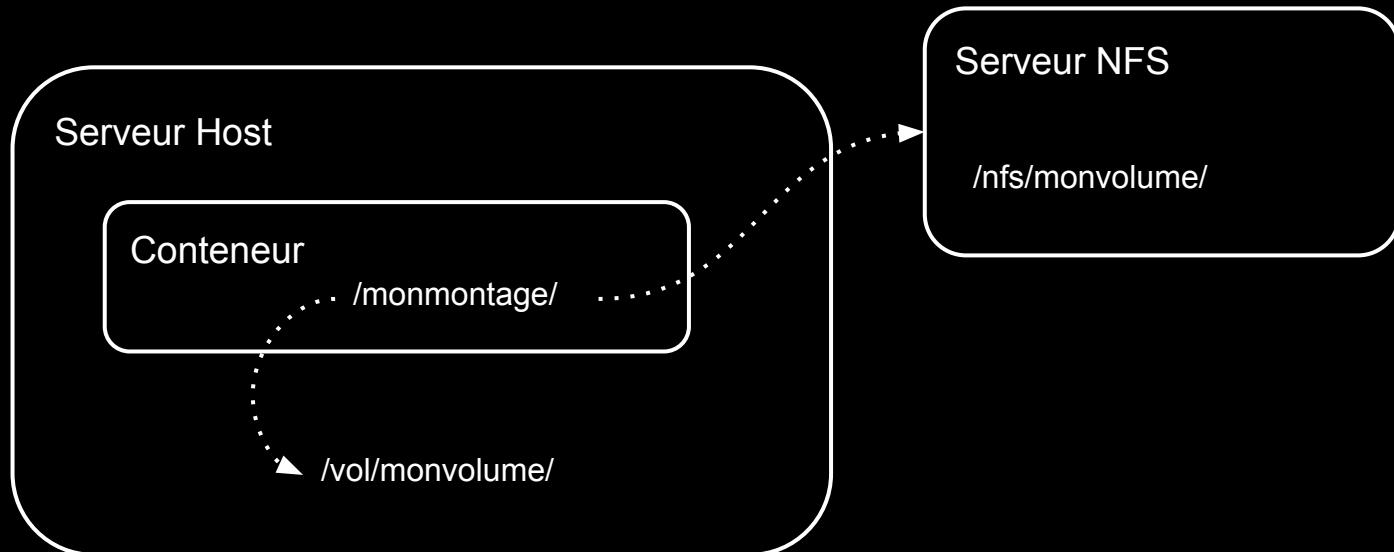
# Rappel sur les conteneurs et Docker

## Docker run

- docker run nginx (-d)
- docker run -d --name monginx nginx
- docker ps (-a)
- docker start/stop monginx (restart)
- docker exec -ti monginx /bin/bash (shell)
- docker inspect monginx
- docker rm monginx (-f)

# Rappel sur les conteneurs et Docker

## Docker volumes : principes



# Rappel sur les conteneurs et Docker

## Docker volumes : commandes

- docker volume ls
- docker volume create <monvolume>
- docker volume rm <monvolume>
- docker run --name monconteneur -v /vol/monvolume:/monmontage nginx
- docker run --name monconteneur -v labelmonvolume:/monmontage nginx
- docker volume create --driver local --opt type=nfs --opt o=addr=<ip\_nfs>,rw --opt device=/nfs/monvolume monvolume

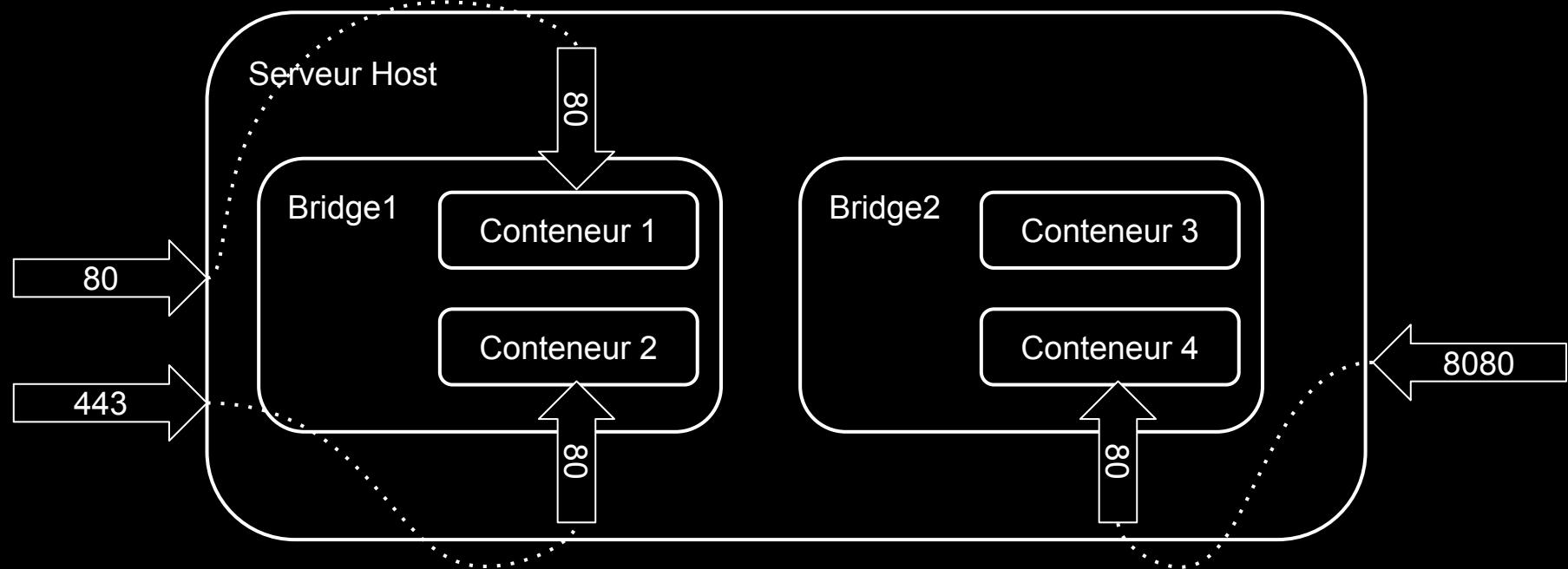
# Rappel sur les conteneurs et Docker

## Docker network : commandes

- docker network ls
- docker network create -d bridge --subnet 172.30.0.0/16 <monnet>
- docker volume rm <monnet>
- docker run --name monconteneur --network <monnet> nginx
- docker run --name monconteneur --network host nginx
- docker ports <monconteneur>
- docker run --name monconteneur -p 8080:80 nginx

# Rappel sur les conteneurs et Docker

## Docker network : principes



# Rappel sur les conteneurs et Docker

## Docker : commandes

- docker run (--name -p...)
- docker ps : liste les conteneurs (-a)
- docker inspect <monconteneur> (ip)
- docker stop <conteneurs>
- docker start <conteneur>
- docker rm <conteneur> (-f)
- docker images (image ls)
- docker image rm <monimage> (rmi)
- docker build -t <nom\_tag\_image> .

# Rappel sur les conteneurs et Docker

## Docker Compose

- compose = orchestrateur de conteneurs
- fichier descriptif format yaml
- gestion idempotence (ne refait pas les choses déjà faites)
- scaling facilité (+/- swarm)
- binaire à installer à part
- sudo curl -L

```
"https://github.com/docker/compose/releases/download/1.25.4/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

# Rappel sur les conteneurs et Docker

## Docker Compose

Compose file format	Docker Engine release
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+
1.0	1.9.1.+

# Rappel sur les conteneurs et Docker

## Docker Compose

Documentation : <https://docs.docker.com/compose/compose-file/>

Structure : version / services / nom\_services

```
version: "3.7"
services:
  webapp:
    build: ./dir
```

Build :

```
version: "3.7"
services:
  webapp:
    build:
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        buildno: 1
```

# Rappel sur les conteneurs et Docker

## Docker Compose

image : image utilisé

container\_name : nom du conteneur

depends\_on : service préalable

volumes : liste des volumes

networks : liste des réseaux

environment : définir la liste des variables d'environnement

healthcheck : modalité de vérification de l'état du conteneur (retries, curl...)

restart : modalités pour relancer un conteneur

command : commande passée au conteneur (si autre que le entrypoint)

# Rappel sur les conteneurs et Docker

## Docker Compose

```
version: '3.3'

services:
  db:
    container_name: mysql
    image: mysql:5.7
    volumes:
      - wp_db:/var/lib/mysql/
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress
    networks:
      - wp

networks:
  wp:

volumes:
  wp_db:
    driver: local
    driver_opts:
      o: bind
      type: none
      device: /srv/wordpress/db
```

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

# Script d'environnement BDD

## Objectif

- pouvoir faire des tests unitaires en lien avec des outils/technos (postgres par exemple)
- lancer facilement une instance (bdd, kafka etc)
- supprimer rapidement une fois les tests réalisés
- sans laisser de traces
- automatisation

=> Conteneurs => docker (au choix) + docker-compose

# Script d'environnement BDD

## Pourquoi un script ?

- on aurait pu ajouter un docker compose dans un dépôt (applicatif ?)
- on aurait pu lancer un simple docker run
- mais on peut capitaliser dans ce script (l'emporter partout avec vous : home...)

# Script d'environnement BDD

## Script

- entête : aide
- entête : parser d'options
- fonctions
- exécution

```
#!/bin/bash

DIR="${HOME}/generator"
USER_SCRIPT=$USER

# Fonctions #####
# help_list() {
#   echo "Usage:
# ${0##*/} [-h][--prometheus][--grafana]

Options:
-h, --help
  can I help you ?

-c, --cassandra
  run cassandra container

-a, --api
  run api for test

-i, --ip
  list ip for each container

-pro, --prometheus
  run prometheus container

-gra, --grafana
  run grafana container

-j, --jenkins
  run jenkins container

-pos, --postgres
  run postgres

-m, --mariadb
  run mariadb

-gil, --gitlab
  run gitlab

"
}
```

# Script d'environnement BDD

## Script

- entête : aide
- entête : parser d'options
- fonctions
- exécution

```
parse_options() {
    case $@ in
        -h|help)
            help_list
            exit
        ;;
        -a|--api)
            api
            ;;
        -c|--cassandra)
            cassandra
            ;;
        -pro|--prometheus)
            prometheus
            ;;
        -gra|--grafana)
            grafana
            ;;
        -i|--ip)
            ip
            ;;
        -m|--mariadb)
            mariadb
            ;;
        -pos|--postgres)
            postgres
            ;;
        -j|--jenkins)
            jenkins
            ;;
        -gil|--gitlab)
            gitlab
            ;;
        *)
            echo "Unknown option: ${opt} - Run ${0##*/} -h for help." >&2
            exit 1
    esac
}
```

# Script d'environnement BDD

## Script

- entête : aide
- entête : parser d'options
- fonctions
- exécution

```
ip() {
for i in $(docker ps -q); do docker inspect -f "{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}} - {{.Name}}" $i;done
}
```

# Script d'environnement BDD

## Script

- entête : aide
- entête : parser d'options
- fonctions
- exécution

```
postgres() {  
    echo  
    echo "Install Postgres"  
  
    echo "1 - Create directories ${DIR}/generator/postgres/"  
    mkdir -p $DIR/postgres/  
  
    echo "  
version: '3.0'  
services:  
  web:  
    image: postgres:latest  
    container_name: postgres  
    environment:  
      - POSTGRES_USER=myuser  
      - POSTGRES_PASSWORD=myuserpassword  
      - POSTGRES_DB=mydb  
    ports:  
      - 5432:5432  
    volumes:  
      - postgres_data:/var/lib/postgresql/  
    networks:  
      - generator  
volumes:  
  postgres_data:  
    driver: local  
    driver_opts:  
      o: bind  
      type: none  
      device: ${DIR}/postgres  
networks:  
  generator:  
    driver: bridge  
    ipam:  
      config:  
        - subnet: 192.168.168.0/24  
" >$DIR/docker-compose-postgres.yml  
  
echo "2 - Run postgres"  
docker-compose -f $DIR/docker-compose-postgres.yml up -d  
  
echo "  
Credentials:  
  user: myuser  
  password: myuserpassword  
  db: mydb  
  port: 5432  
  
command : psql -h <ip> -u myuser mydb  
"  
}
```

# Script d'environnement BDD

## Script

- entête : aide
- entête : parser d'options
- fonctions
- exécution

```
# Let's Go !! parse args #####
parse_options $@
ip
(END)
```

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

11- Jenkins : introduction

# Jenkins

## Job

- Hello World en shell
- Hello World en pipeline

Pipeline

Definition Pipeline script

Script	1 node{ 2 try{ 3 stage('Great') { 4 sh 'echo Hello World' 5 } 6 finally { 7 cleanWs() 8 } 9 }	?
--------	-----------------------------------------------------------------------------------------------------------------------	---

# Jenkins

## Plugin Multibranch Pipeline

- Hello World en pipeline

The screenshot shows the Jenkins Multibranch Pipeline interface for the 'MultiHello' project. At the top, there is a folder icon followed by the project name 'MultiHello'. Below this, a 'Branches (2)' section is displayed, showing two branches: 'dev' and 'master'. The 'dev' branch was last successful 15 seconds ago, while the 'master' branch was last successful 1 minute 24 seconds ago. Both branches have a duration of approximately 0.61 seconds. Each branch entry includes a small circular icon with a blue sphere and a yellow sun, and a green circular icon with a white globe.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		<a href="#">dev</a>	15 sec - #1	N/A	0.61 sec
		<a href="#">master</a>	1 min 24 sec - #1	N/A	0.77 sec

Icon: [S](#) [M](#) [L](#) [Legend](#) Atom feed for all Atom feed for failures Atom feed for just latest builds

# Jenkins

## Trigger Gitab

**Build Triggers**

Build after other projects are built 

Build periodically 

Build when a change is pushed to GitLab. GitLab webhook URL: <http://192.168.5.2:8080/project/MultiPipeline/master> 

Enabled GitLab triggers	
Push Events	<input checked="" type="checkbox"/>
Opened Merge Request Events	<input checked="" type="checkbox"/>
Accepted Merge Request Events	<input type="checkbox"/>
Closed Merge Request Events	<input type="checkbox"/>
Rebuild open Merge Requests	Never 
Approved Merge Requests (EE-only)	<input checked="" type="checkbox"/>
Comments	<input checked="" type="checkbox"/>

Comment (regex) for triggering a build  

[Advanced...](#)

# Jenkins

## Trigger Gitab

The screenshot shows the GitLab Admin Area Network settings page. The left sidebar is titled "Settings" and includes sections for General, Integrations, Repository, CI/CD, Reporting, Metrics and profiling, and Network. The Network section is currently selected and highlighted in grey. The main content area is titled "Network" and contains three sections: "Performance optimization", "User and IP Rate Limits", and "Outbound requests". The "Outbound requests" section is expanded, showing a "Whitelist to allow requests to the local network from hooks and services" input field containing the IP address "192.168.10.2". A note below the input field explains the allowed formats for domain names and IP ranges. At the bottom of the page is a "Save changes" button.

GitLab Projects Groups More

Monitoring

Messages

System Hooks

Applications

Abuse Reports 0

Kubernetes

Deploy Keys

Service Templates

Labels

Appearance

Settings

General

Integrations

Repository

CI/CD

Reporting

Metrics and profiling

Network

Preferences

Admin Area > Network

Performance optimization

Various settings that affect GitLab performance.

User and IP Rate Limits

Configure limits for web and API requests.

Outbound requests

Allow requests to the local network from hooks and services.

Allow requests to the local network from web hooks and services

Allow requests to the local network from system hooks

Whitelist to allow requests to the local network from hooks and services

192.168.10.2

Requests to these domain(s)/address(es) on the local network will be allowed when local requests from hooks and services are not allowed. IP ranges such as 1.0:0.0:0:0/124 or 127.0.0.0/28 are supported. Domain wildcards are not supported currently. Use comma, semicolon, or newline to separate multiple entries. The whitelist can hold a maximum of 1000 entries. Domains should use IDNA encoding. Ex: example.com, 192.168.1.1, 127.0.0.0/28, xn--itlab-j1a.com.

Enforce DNS rebinding attack protection

Resolves IP addresses once and uses them to submit requests

Save changes

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

11- Jenkins : introduction

12- Maven

# Maven

- gestion de projet java
- gestion de dépendances
- builder le projet
- actions :

- `validate` - validate the project is correct and all necessary information is available
- `compile` - compile the source code of the project
- `test` - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- `package` - take the compiled code and package it in its distributable format, such as a JAR.
- `verify` - run any checks on results of integration tests to ensure quality criteria are met
- `install` - install the package into the local repository, for use as a dependency in other projects locally
- `deploy` - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

# Maven

- installation sur le serveur Jenkins ou slave
- ou via docker (évite les dépendances de versions Java...)

```
stage('SERVICE - Tests unitaires'){
    sh 'docker run --rm --name maven-${commitIdLong} -v /var/lib/jenkins/maven/:/root/.m2 -v
'$(pwd)":/usr/src/mymaven --network generator_generator -w /usr/src/mymaven maven:3.3-jdk-8 mvn -
B clean test'
}

stage('SERVICE - Jar'){
    sh 'docker run --rm --name maven${commitIdLong} -v /var/lib/jenkins/maven/:/root/.m2 -v "
$(pwd)":/usr/src/mymaven --network generator_generator -w /usr/src/mymaven maven:3.3-jdk-8 mvn -B
clean install'
}
```

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

11- Jenkins : introduction

12- Maven

13 - Docker : build et push

# Docker

## Dockerfile

- objectif : créer une image
- à partir d'une autre image (jdk)
- actions :
  - FROM <nom\_image> : image d'origine
  - RUN <commande> : lancer une commande
  - ADD/COPY <source\_host> <dest\_container> : copie d'un fichier/répertoire
  - ENV <variable> <valeur> : définition d'une variable d'environnement
  - WORKDIR <rédertoire> : se rendre dans un répertoire
  - EXPOSE <port> : expose un port
  - ENTRYPOINT <conteneur> : processus du conteneur
  - CMD <args> : paramètres par défaut (liste)
  - USER <user> : user qui fait tourner le processus (à créer)

# Docker

- packager une image à partir d'une jdk
- installer le jar dans l'image (copy)
- pousser l'image vers une registry

```
FROM openjdk:14-slim-buster
CMD mkdir /jar
COPY target/*.jar /jar/
EXPOSE 8080
```

```
'curl -sk --user $USERNAME:$PASSWORD https://192.168.10.5:5000/v2/first-pipeline/tags/list'
```

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

11- Jenkins : introduction

12- Maven

13 - Docker : build et push

14 - Jenkins : assemblage partie 1

# Jenkins

## Jenkinsfile

- Objectif : mettre en place le Jenkinsfile
  - variable d'environnement
  - charger le dépôt git de generator postgres
  - lancer le conteneur postgres
  - clean workspace

# Jenkins

## Job

- Hello World en shell
- Hello World en pipeline

Pipeline

Definition Pipeline script

Script	1 node{ 2 try{ 3 stage('Great') { 4 sh 'echo Hello World' 5 } 6 finally { 7 cleanWs() 8 } 9 }	?
--------	-----------------------------------------------------------------------------------------------------------------------	---

# Jenkins

## Jenkinsfile

- préparation de l'environnement Jenkins
- lancement de postgres en local via le script

```
def buildNum = env.BUILD_NUMBER
def branchName= env.BRANCH_NAME

stage('POSTGRESQL - Git checkout'){
git 'http://gitlab.example.com/root/pipeline-docker-generator.git'
}

stage('POSTGRESQL - Container run'){
sh "./generator.sh -pos"
sh "docker ps -a"
}
```

# Jenkins

## Jenkinsfile

- git clone
- définition de variables

```
stage('SERVICE - Git checkout'){
    git branch: branchName, url: 'http://gitlab.example.com/root/pipeline-myapp1.git'}

/* Variables */
def imageName='192.168.10.5:5000/first-pipeline'

if (branchName == "dev" ){
    extension = "-SNAPSHOT"
}
if (branchName == "stage" ){
    extension = "-RC"
}
if (branchName == "master" ){
    extension = ""
}
```

# Jenkins

## Jenkinsfile

- versionning

```
def commitIdLong = sh returnStdout: true, script: 'git rev-parse HEAD'
def commitId = commitIdLong.take(7)
sh "sed -i s/'-XXX'/${extension}/g pom.xml"
def version = sh returnStdout: true, script: "cat pom.xml | grep -A1 '<artifactId>myapp1' | tail -1 | perl -nle '\n{.*<version>(.*)</version>.*};print \$1' | tr -d '\n'"
print """
#####
BranchName: $branchName
CommitID: $commitId
AppVersion: $version
JobNumber: $buildNum
ImageName: $imageName
#####
"""

```

# Jenkins

## Jenkinsfile

- maven : build/tests

```
stage('SERVICE - Tests unitaires'){
    sh 'docker run --rm --name maven-${commitIdLong} -v /var/lib/jenkins/m
aven/:/root/.m2 -v "$(pwd)":/usr/src/mymaven --network generator_generator -w
/usr/src/mymaven maven:3.3-jdk-8 mvn -B clean test'
}

stage('SERVICE - Jar'){
    sh 'docker run --rm --name maven${commitIdLong} -v /var/lib/jenkins/ma
ven/:/root/.m2 -v "$(pwd)":/usr/src/mymaven --network generator_generator -w /
usr/src/mymaven maven:3.3-jdk-8 mvn -B clean install'
}
```

# Jenkins

## Jenkinsfile

- docker : build/push

```
stage('DOCKER - Build/Push registry'){
    print "$imageName:${version}-${commitId}"
    docker.withRegistry('http://192.168.10.5:5000', 'myregistry_login'
) {
    def customImage = docker.build("$imageName:${version}-${commitId}")
    customImage.push()
}
sh "docker rmi $imageName:${version}-${commitId}"
}
```

# Jenkins

## Jenkinsfile

- docker : check ?

```
stage('DOCKER - check registry'){
    withCredentials([[${class: 'UsernamePasswordMultiBinding', credentialsId: 'myregistry_login',usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD'}]]) {
        sh 'curl -sk --user $USERNAME:$PASSWORD https://192.168.10.5:5000/v2/first-pipeline/tags/list'
    }
}
```

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

11- Jenkins : introduction

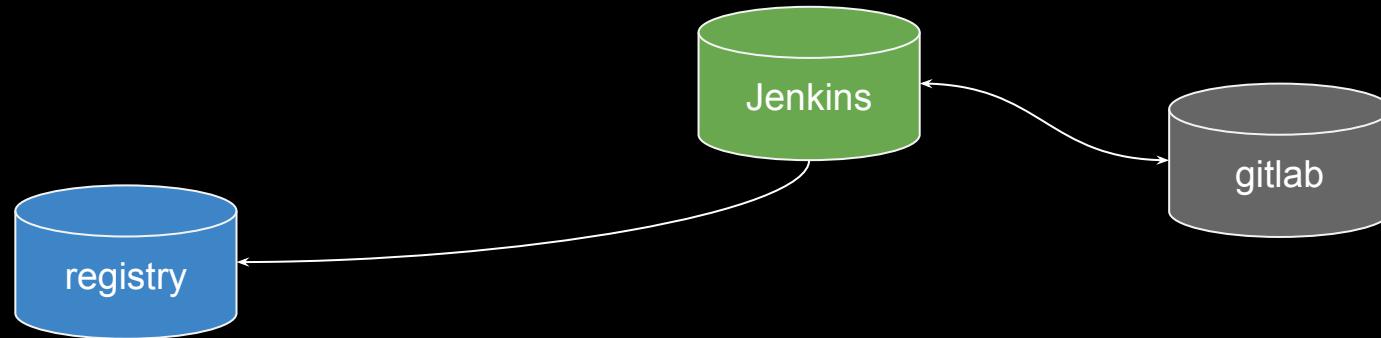
12- Maven

13 - Docker : build et push

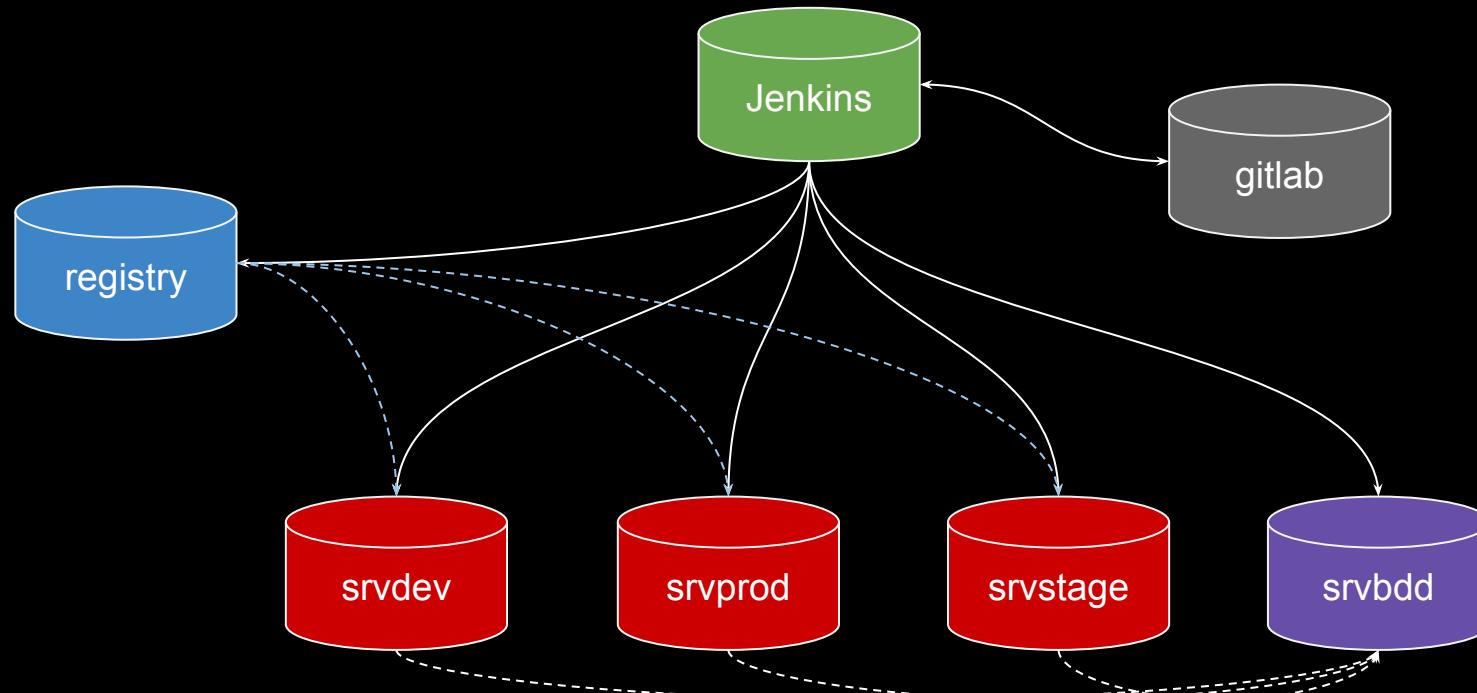
14 - Jenkins : assemblage partie 1

15 - Ansible : introduction

# Ansible : déploiement

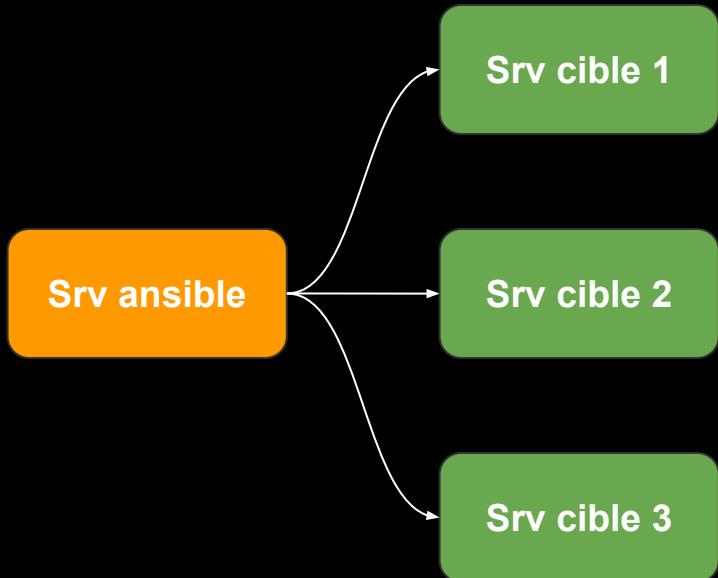


# Ansible : déploiement



# Ansible : déploiement

- python
- déploiement/installation à distance via SSH
- permet :
  - installation d'infrastructure as code
  - déploiement type pipeline



# Ansible : déploiement

## Définitions

- Inventory : liste des serveurs et variables
- Tâche : une action à réaliser
- Module : fonctions appelées par des tâches
- Rôle : regroupement de tâches visant à déployer/installer un bloc spécifique cohérent
- Playbook : Définition des rôles devant être joués sur quels groupes de serveurs (inventory)
- Groupes : division de regroupant des serveurs par catégorie au sein de l'inventory
- Variables

# Ansible : déploiement

## Inventory

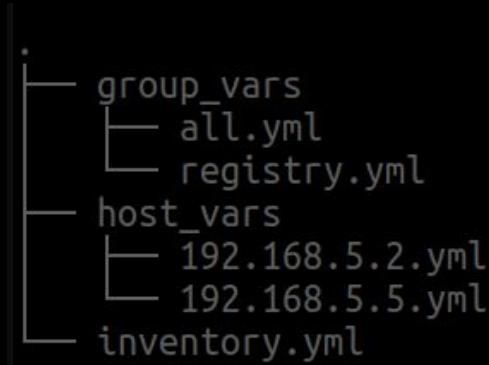
- Inventory : liste des serveurs et variables
- Divisé en groupes (all / children)
- définition de variable

```
all:  
  vars:  
    nom: "xavki"  
  children:  
    jenkins:  
      hosts:  
        192.168.5.2:  
    registry:  
      hosts:  
        192.168.5.5:
```

# Ansible : déploiement

## Group et Host Vars

- Hiérarchie des variables (22)
- Fichiers yaml spécifiques // inventory
- Définition de variable



# Ansible : déploiement

## Exemples :

- ansible -i inventory.yml all -u vagrant -k -m ping
- ansible -i inventory.yml all -u vagrant -k -m shell -a "hostname"
- ansible -i inventory.yml all -u vagrant -k -m shell -a "echo 'yo' > /tmp/xp"
- ansible -i inventory.yml jenkins -u vagrant -k -m shell -a "echo 'yo' > /tmp/xp"
- ansible -i inventory.yml jenkins -u vagrant -k -m shell -a "echo '{{nom}}' > /tmp/xp"
- ansible -i inventory.yml 192.168.5.5 -u vagrant -k -m shell -a "echo '{{nom}}' > /tmp/xp"

# Ansible : déploiement

## Playbook

- association entre
  - inventory (groups, hosts...)
  - rôles
  - tâches sans rôles

```
- name: premier playbook
hosts: jenkins
tasks:
- name: première tâche
  shell: echo "toto" > /tmp/xp
roles:
- test
```

# Ansible : déploiement

## Rôles

- ensemble de tâche
- organisé en répertoire
  - tasks : tâches
  - defaults : variables par défaut
  - vars : variables non modifiables
  - files : fichiers
  - templates : modèles de fichiers
  - handlers : déclencheurs

```
(master *) $ ▶ tree roles/test/
roles/test/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
└── tests
    ├── inventory
    └── test.yml
└── vars
    └── main.yml
```

# Ansible : déploiement

## Roles/Files

- répertoire files ou pas
- fichiers “statiques”
- module copy
  - src : source
  - dest : destination
  - owner / group
  - mode

```
- name: task 1
copy:
  src: monfichier.txt
  dest: /tmp/xavki.txt
  mode: 0755
  owner: vagrant
```

# Ansible : déploiement

## Roles/Templates

- répertoire templates ou pas
- fichiers “dynamiques” jinja (python)
- module template
  - src : source
  - dest : destination
  - owner / group
  - mode

```
- name: task 1
  template:
    src: montemplate.txt.j2
    dest: /tmp/xavki.txt
    mode: 0755
    owner: vagrant
```

# Ansible : déploiement

## Roles/Handlers/Apt

- module apt : gestion paquets
- module template + notify
  - déclencheur
  - sur modification (spécifique à templates)
  - joue une tâche appelée par son nom

```
- name: task 1
become: yes
apt:
  name: nginx
  state: present

- name: changement de configuration
become: yes
template:
  src: default.conf.j2
  dest: /etc/nginx/sites-available/default
notify: reload_nginx
```

# Ansible : déploiement

## Roles/Handlers/Apt

- handlers : liste de tâches
- attendent un trigger

```
- name: reload_nginx
become: yes
service:
  name: nginx
  state: reloaded
```

# Ansible : déploiement

## Pull Roles

- ansible galaxy
- dépôts git
- fichiers requirements
- ansible-galaxy install --roles-path roles -r requirements.yml
- 3 rôles :
  - docker/docker compose
  - installation applicatif
  - installation base de données

```
- src: https://github.com/ultransible/docker.git
  name: docker
  version: master
  scm: git
```

```
- name: playbook 1
  hosts: stage
  become: yes
  roles:
    #- monrole
    - docker
```

# Ansible : déploiement

## Pull Roles

- modification inventory
- modification playbook > docker
- variables group\_vars/all.yml
  - remap user docker
  - utilisation registry insecure

```
all:  
  children:  
    jenkins:  
      hosts:  
        192.168.5.2:  
    registry:  
      hosts:  
        192.168.5.5:  
    stage:  
      hosts:  
        192.168.5.7:
```

```
- name: playbook 1  
  hosts: stage  
  become: yes  
  roles:  
    #- monrole  
    - docker
```

```
docker_docker_conf:  
  userns-remap: "dockremap:dockremap"  
  insecure-registries: ["192.168.5.5:5000"]
```

# Ansible : déploiement

## Roles Postgres

- objectif ?

- postgres
- pip
- psycopg2 (pip)

```
- name: Enable passwordless sudo
  become: yes
  lineinfile: dest=/etc/sudoers regexp='^vagrant' line="vagrant ALL=(postgres) NOPASSWD:/bin/sh"

- name: install pip
  become: yes
  apt:
    name:
      - postgresql-11
      - python-pip
      - postgresql-server-dev-11
    state: present
    update_cache: yes
    cache_valid_time: 3600

- name: install pip psycopg2
  become: yes
  pip:
    name: psycopg2
```

# Ansible : déploiement

## Roles Postgres

- installation user
  - vagrant/vagrant (/!\ password)
- installation database
  - une par environnement

```
[ssh_connection]
pipelining=True

[defaults]
allow_world_readable_tmpfiles = True
```

```
pg_settings:
  db_name:
    - { name: "formation", owner: "vagrant" }
  users:
    - { name: "vagrant", password: "vagrant" }
```

```
- name: create user
  become_user: postgres
  postgresql_user:
    name: "{{ item.name }}"
    password: "{{ item.password }}"
    encrypted: true
  with_items: "{{ pg_settings.users }}"
  when: pg_settings.users is defined

- name: create database
  become_user: postgres
  postgresql_db:
    name: "{{ item.name }}"
    owner: "{{ item.owner }}"
  with_items: "{{ pg_settings.db_name }}"
  when: pg_settings.db_name is defined
```

# Ansible : déploiement

## Roles Applicatif

- installer une image docker
  - login registry
  - compose
- installer la configuration
- run docker compose
- attente réponse

# Ansible : déploiement

## Roles Applicatif

- installation application.properties

```
- name: create main directories
  file:
    dest: /xavki/etc/
    mode: 0775
    recurse: yes
    state: directory

- name: insert config file
  template:
    src: application.properties.j2
    dest: /xavki/etc/application.properties
    mode: 0755
```

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url={{ service_install_run_data_source_connection }}
spring.datasource.username={{ service_install_run_data_source_connection_user }}
spring.datasource.password={{ service_install_run_data_source_connection_password }}

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

# Ansible : déploiement

## Roles Applicatif

- docker-compose

```
service_install_run_data_source_connection: "jdbc:postgresql://192.168.5.6:5432/formation"
service_install_run_data_source_connection_user: "vagrant"
service_install_run_data_source_connection_password: "vagrant"

service_install_run_registry_url: "https://192.168.5.5:5000"
service_install_run_registry_user: "xavki"
service_install_run_registry_password: "password"

service_install_run_image: "192.168.5.5:5000/myapp:0.0.1-SNAPSHOT-06226f3"
service_install_run_version: "0.0.1-SNAPSHOT"
```

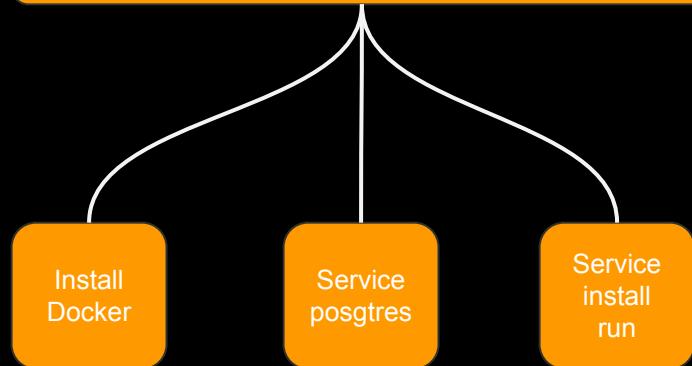
```
version: '3.0'
services:
  myapp1:
    image: {{ service_install_run_image }}
    container_name: myapp1
    working_dir: /xavki/etc
    command: java -jar /jar/myapp1-{{ service_install_run_version }}.jar --spring.config.location=file:/etc/application.properties
    volumes:
    - /xavki/etc/application.properties:/etc/application.properties
    ports:
    - 8080:8080
```

# Ansible : déploiement

## Organisation

- dépôts git des rôles :
  - postgres
  - docker/compose
  - installation service
- multi environnements > un dépôt regroupant
  - playbook
  - inventory
  - variables

Deploy ansible : inventory + vars



# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

11- Jenkins : introduction

12- Maven

13 - Docker : build et push

14 - Jenkins : assemblage partie 1

15 - Ansible : introduction

16 - Jenkins : assemblage partie 2

# Jenkins : assemblage part 2

## Ansible

- SSH : user jenkins > vagrant
- installation plugin jenkins pour ansible
- ajout du code groovy

```
stage('ANSIBLE - Deploy'){
    git branch: 'master', url: 'http://gitlab.example.com/mypipeline/deploy-ansible.git'
    sh "mkdir roles"
    sh "ansible-galaxy install --roles-path roles -r requirements.yml"
    ansiblePlaybook (
        colorized: true,
        playbook: "install-myapp1.yml",
        credentialsId: 'ssh_vagrant',
        hostKeyChecking: false,
        inventory: "env/${branchName}/hosts",
        extras: "-u vagrant -e 'imageName=${imageName}-${commitId}' -e 'version=${version}'"
    )
}
```

# Jenkins : assemblage part 2

## Script Generator

- éviter les conflits de conteneurs
  - passage d'un id pour le nom
  - pas d'exposition de port (!\ ip)
- pas de stockage persistent
- clean en fin de job
- clean des volumes

```
optspec=":ihv-:"  
while getopts "$optspec" optchar; do  
    case "${optchar}" in  
        -)  
            case "${OPTARG}" in  
                postgres)  
                    arg="${!OPTIND}"; OPTIND=$(( $OPTIND + 1 ))  
                    postgres "$arg"  
                    ;;  
                clean)  
                    arg="${!OPTIND}"; OPTIND=$(( $OPTIND + 1 ))  
                    clean "$arg"  
                    ;;  
                ip)  
                    ip ;;  
                help) echo "Erreur reportez-vous à l'aide"  
                    help_list ;;  
                *) echo "Erreur reportez-vous à l'aide"  
                    help_list ;;  
            esac;  
        i) ip ;;  
        p) postgres ;;  
        h) help_list ;;  
        *) echo "Erreur reportez-vous à l'aide"  
            help_list ;;  
    esac  
done  
~
```

# Jenkins : assemblage part 2

## Jenkinsfile

- ajustement suite aux évolutions
- passage d'un id au script
- récupération de l'ip
- modification applications.properties (/!\ ip)
- ajout d'un clean final

```
stage('Env - run postgres'){
    sh "./generator.sh --postgres pipeline${buildNum}${branchName}"
    sh "docker ps -a"
}
def ipPostgres = sh returnStdout: true, script: "./generator.sh -i | grep postgrespipeline${buildNum}${branchName} | awk '{print \$1}' | tr -d '\n'"
```

```
/* Ajout du nom du conteneur dans application properties */
sh "sed -i s/'XXX'/${ipPostgres}/g src/main/resources/application.properties"
```

```
git "http://gitlab.example.com/mypipeline/generator.git"
sh "./generator.sh --clean postgrespipeline${buildNum}${branchName}"
```

# Jenkins : assemblage part 2

## Jenkinsfile > Ansible

- passage de variables
  - image
  - version

```
version: '3.0'
services:
  myapp1:
    image: {{ image }}
    container_name: myapp1
    working_dir: /xavki/etc
    command: java -jar /jar/myapp1-{{ version }}.jar --spring.config.location
              =file:/etc/application.properties
    volumes:
      - /xavki/etc/application.properties:/etc/application.properties
    ports:
      - 8080:8080
```

# Jenkins : assemblage part 2

## Trigger Gitlab

- déclenchement par branche
- ajout des triggers
- modification conf jenkinsfile

**Edit Project Hook**

Project Hooks can be used for binding events when something is happening within the project.

**URL**  
http://192.168.5.2:8080/project/DeployMyApp/dev

**Secret Token**  
abcdefghijklmnoprstuvwxyz0123456789ABCDEF

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-T

**Trigger**

**Push events**  
This URL will be triggered by a push to the repository  
dev

**Tag push events**  
This URL will be triggered when a new tag is pushed to the repository

**Comments**

```
def project_token = 'abcdefghijklmnoprstuvwxyz0123456789ABCDEF'
def branchName= env.BRANCH_NAME
def buildNum = env.BUILD_NUMBER

properties([
    gitLabConnection('your-gitlab-connection-name'),
    pipelineTriggers([
        [
            $class: 'GitLabPushTrigger',
            branchFilterType: 'All',
            triggerOnPush: true,
            triggerOnMergeRequest: true,
            triggerOpenMergeRequestOnPush: "never",
            triggerOnNoteRequest: true,
            noteRegex: "Jenkins please retry a build",
            skipWorkInProgressMergeRequest: true,
            secretToken: project_token,
            ciSkip: false,
            setBuildDescription: true,
            addNoteOnMergeRequest: true,
            addCiMessage: true,
            addVoteOnMergeRequest: true,
            acceptMergeRequestOnSuccess: true,
            branchFilterType: "NameBasedFilter",
            includeBranchesSpec: "${branchName}",
            excludeBranchesSpec: ""
        ]
    ])
])
```

# Sommaire

1- Objectifs

2- Devops : pkoi, agile, failfast...

3- Définitions et Outils

4- Infrastructure : vagrant...

5- Applicatif : choix

6- Pipeline : ébauche

7- Git : cli, gitflow

8- Gitlab : présentation, dépôts

9- Docker rappels

10- Script d'environnement (BDD)

11- Jenkins : introduction

12- Maven

13 - Docker : build et push

14 - Jenkins : assemblage partie 1

15 - Ansible : introduction

16 - Jenkins : assemblage partie 2

17 - Jmeter : tests fonctionnels et charges

# Jmeter : tests fonctionnels et charges

## Tests

- simuler une utilisation
- amont = connaître ses utilisateurs / charge cible
- test de charge : lectures et écritures
- tests fonctionnels : lectures et écritures
- peu poussés (prise en compte des risques, caractère spéciaux...)
- check :
  - retour : code http + data
  - base de données
- TESTS = personnalisation maximum
- Important : monitoring (cpu, mémoire, disque...)

# Jmeter : tests fonctionnels et charges

## Jmeter

- n'est pas un service => binaire
- GUI = préparation du plan de test
- CLI : run du plan format jmx
- plugin jenkins : intégration résultat
- beaucoup de fonctionnalités :base de données, java, http, mail, ftp...

# Jmeter : tests fonctionnels et charges

## Jmeter

- Définitions :
  - Thread Group : utilisateurs (nombre...)
  - Sampler : requêtes
  - Listener : valorisation des résultats (graphiques, tableaux...)
  - Logic Controller : coordination des samplers
  - Assertions : contrôle des résultats
  - Timers : gestion des temps de pause
  - Config Element : additif de configuration des samplers (csv, cookies, datas...)
  - Post Processor : agir avec la réponse d'un sampler (regex...)
  - Pre Processor : action avant les samplers

# Jmeter : tests fonctionnels et charges

## Des requêtes...

- GET http://192.168.5.3:8080/questions
- GET <http://192.168.5.3:8080/questions?sort=createdAt.desc>
- "Content-Type: application/json" -X POST -d '{"title":"mon titre","description":"ma description"}' <http://localhost:8080/questions>
- "Content-Type: application/json" -X POST -d '{"text": "voici ma réponse"}'  
<http://localhost:8080/questions/1000/answers>

# Jmeter : tests fonctionnels et charges

## Premier test...

- Installation Jmeter :

```
wget https://downloads.apache.org/jmeter/binaries/apache-jmeter-5.2.1.zip
unzip apache-jmeter-5.2.1.zip -d /usr/bin/jmeter
/usr/bin/jmeter/apache-jmeter-5.2.1/bin/jmeter
```
- Thread Group : nb users / nb de boucles
- Sampler : HTTP Request, GET <http://192.168.5.3:8080/questions>
- Listener : Tree + Table
- Sampler : HTTP Request, POST <http://192.168.5.3:8080/questions>
  - {"title": "mon titre", "description": "mon texte de description"}
- Config element : Header Content-Type = application/json

# Jmeter : tests fonctionnels et charges

## Premier test...

The screenshot shows the JMeter 5.2.1 interface with the following details:

- Toolbar:** Includes icons for File, Edit, Search, Run, Options, Tools, Help, and various test elements like Thread Group, HTTP Request, and Assertions.
- MenuBar:** File, Edit, Search, Run, Options, Tools, Help.
- Status Bar:** 00:00:00, 0 errors, 0 warnings, 0 failures.
- Left Sidebar:** Tree view showing "Test Plan" and "Thread Group". Under "Thread Group", there are items: HTTP Header Manager, GET Questions, POST Questions, and View Results Tree.
- Central Panel - Thread Group Configuration:**
  - Name:** Thread Group
  - Comments:** Action to be taken after a Sampler error:
    - Continue
    - Start Next Thread Loop
    - Stop Thread
    - Stop Test
    - Stop Test Now
  - Thread Properties:**
    - Number of Threads (users): 1
    - Ramp-up period (seconds): 1
    - Loop Count:  Infinite 1
    - Same user on each iteration
    - Delay Thread creation until needed
    - Specify Thread lifetime
    - Duration (seconds): [empty]
    - Startup delay (seconds): [empty]

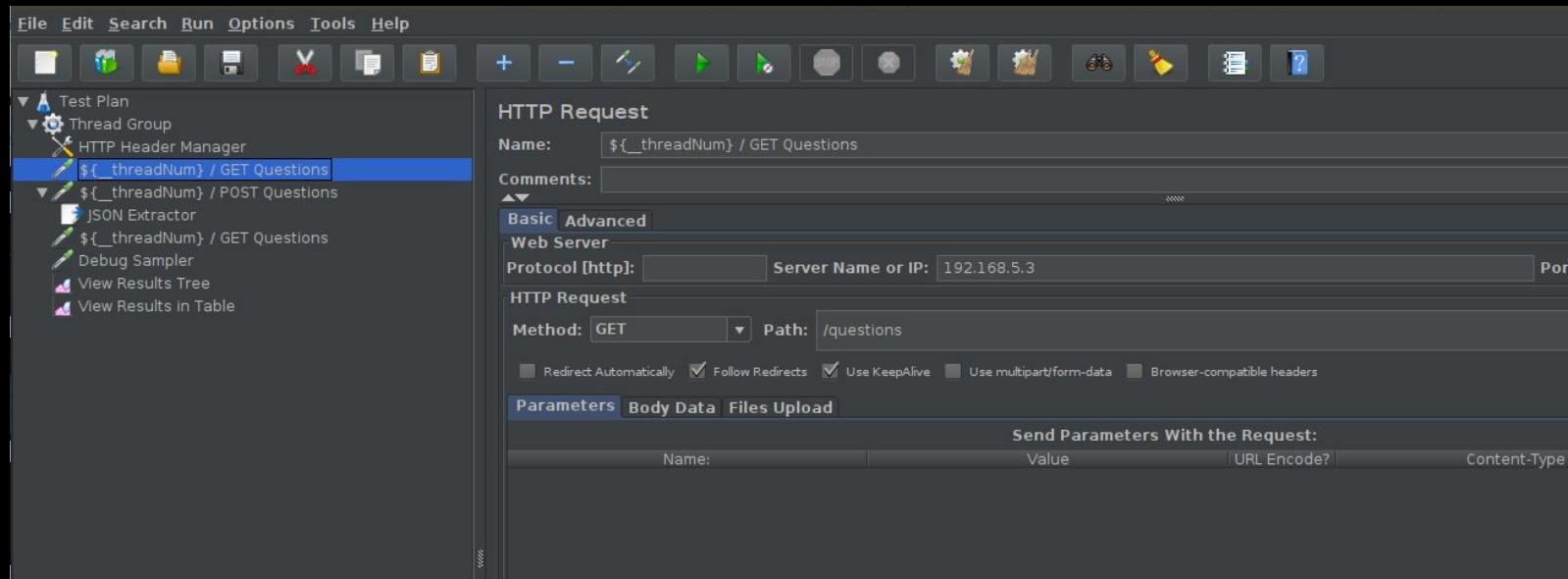
# Jmeter : tests fonctionnels et charges

## Variables, extractor, debug

- \${\_\_threadNum} : variable Id du Thread Group
- Nom de sampler + variations de contenus
- JSON Extractor : sur la réponse du POST
- récupération \$.id ( . : niveau dans la structure)

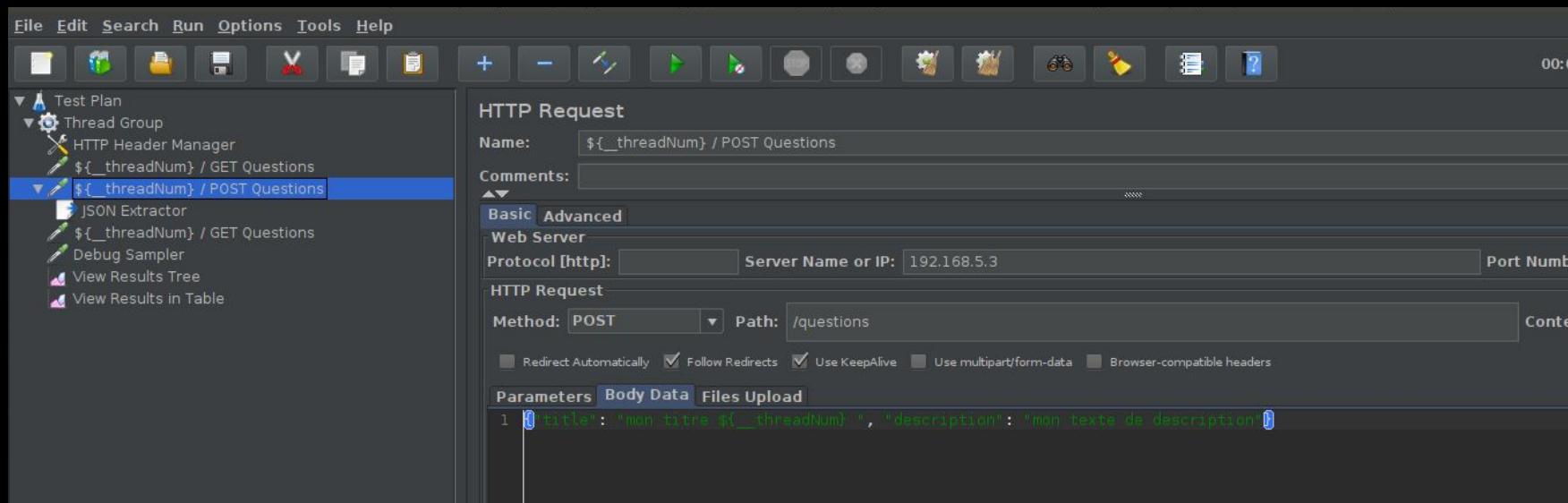
# Jmeter : tests fonctionnels et charges

## Variables, extractor, debug



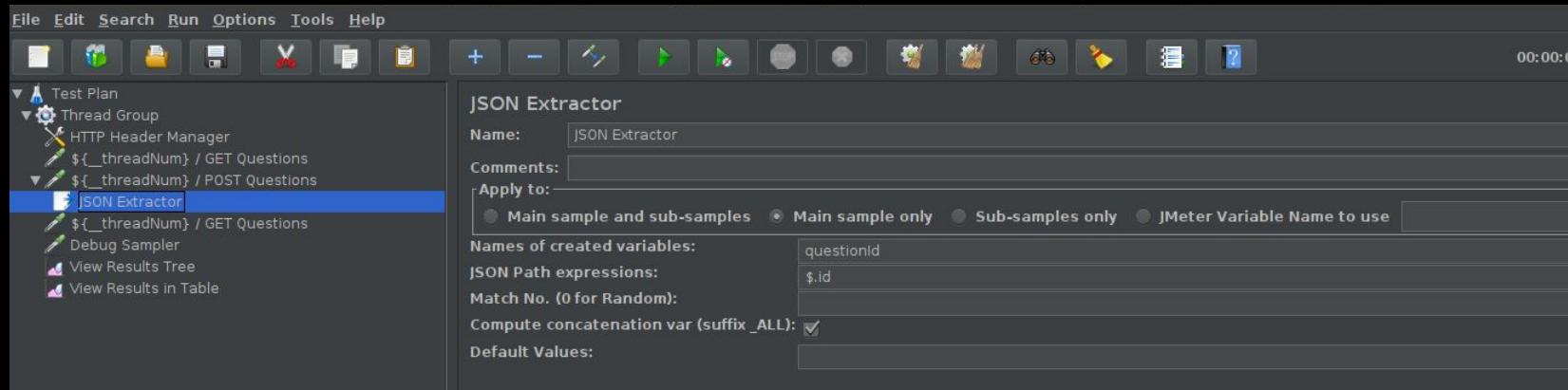
# Jmeter : tests fonctionnels et charges

## Variables, extractor, debug



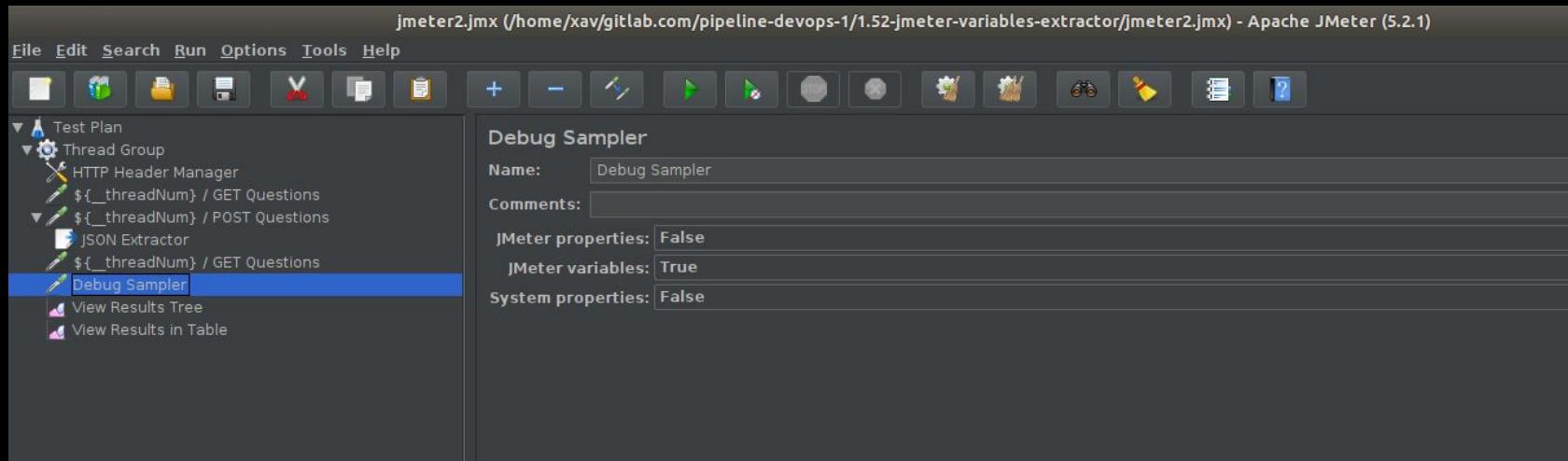
# Jmeter : tests fonctionnels et charges

## Variables, extractor, debug



# Jmeter : tests fonctionnels et charges

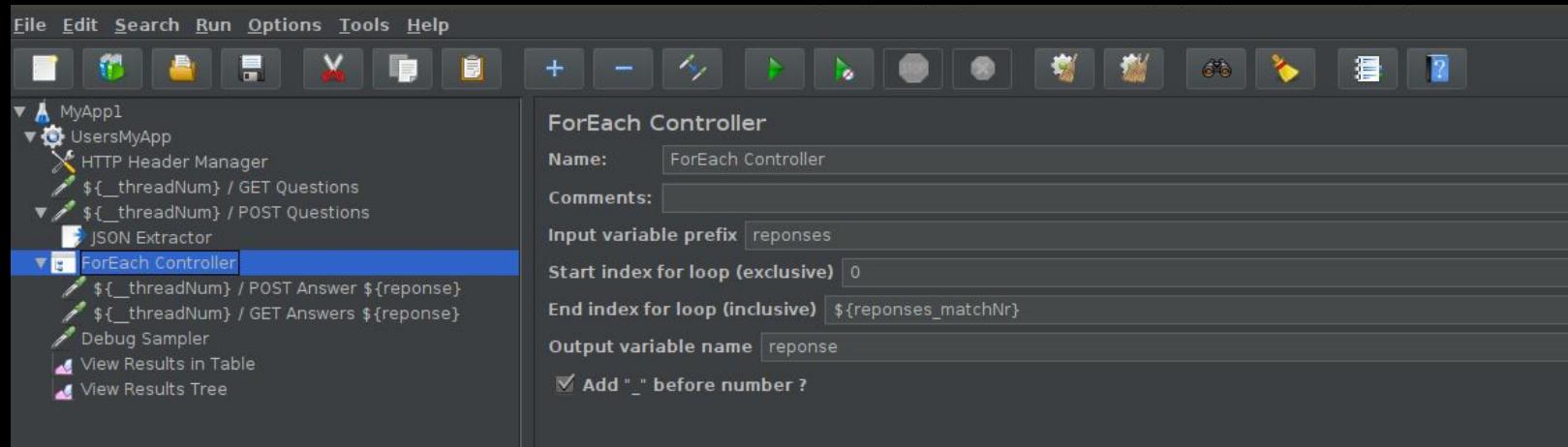
## Variables, extractor, debug



# Jmeter : tests fonctionnels et charges

## For Each et utilisation de variables

- boucle sur une variable liste > n variables simples
- utilisation de la variable de JSON Extractor



# Jmeter : tests fonctionnels et charges

## For Each et Loop Controller

- insertion d'une réponse à chaque question (ou plusieurs)

The screenshot shows the JMeter graphical user interface. The menu bar includes File, Edit, Search, Run, Options, Tools, and Help. The toolbar contains various icons for managing tests, such as creating threads, adding samplers, and viewing results.

The left sidebar displays the test plan structure under the root node 'MyApp'. It includes:

- HTTP Header Manager
- \$\_{threadNum} / GET Questions
- \$\_{threadNum} / POST Questions
- JSON Extractor
- ForEach Controller (selected)
- \$\_{threadNum} / POST Answer \${reponse}
- \$\_{threadNum} / GET Answers \${reponse}
- Debug Sampler
- View Results in Table
- View Results Tree

The main panel shows the configuration for the selected 'ForEach Controller' sampler. The 'HTTP Request' configuration is as follows:

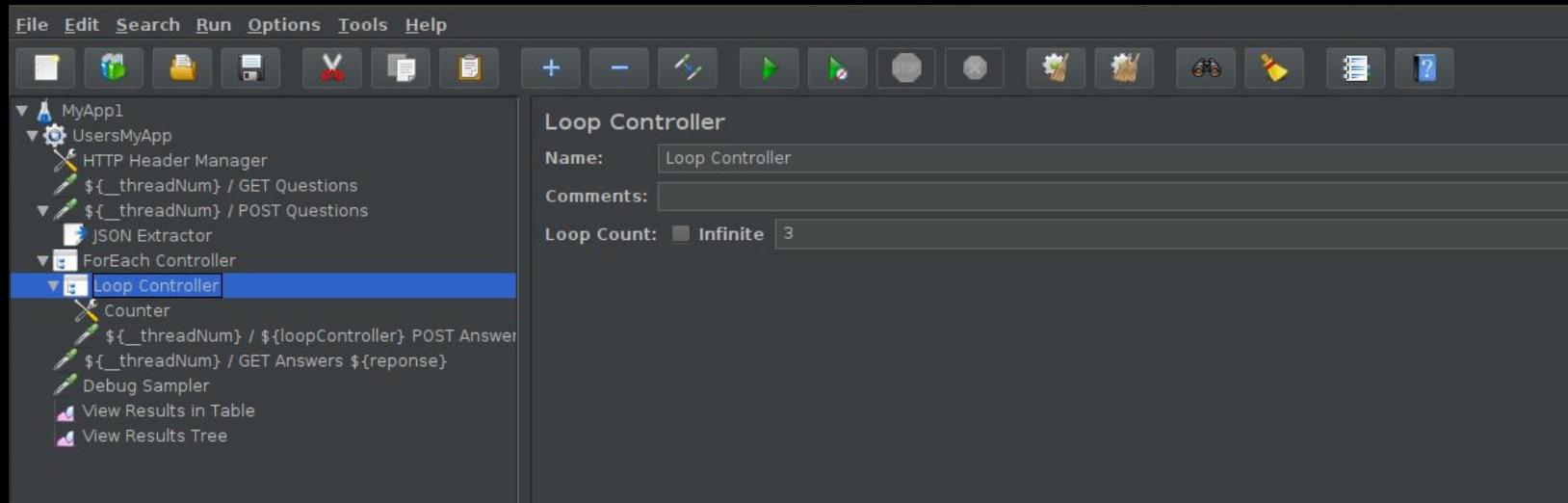
- Name: \${\_threadNum} / POST Answer \${reponse}
- Comments: (empty)
- Protocol [http]: (empty)
- Server Name or IP: 192.168.5.3
- Method: POST
- Path: /questions/\${reponse}/answers
- Parameters tab (selected):
  - 1: "text": "voici la solution"
  - 2: (empty)
  - 3: (empty)
- Body Data tab (disabled): (empty)
- Files Upload tab (disabled): (empty)

Checkboxes at the bottom of the 'HTTP Request' panel include: Redirect Automatically, Follow Redirects, Use KeepAlive, Use multipart/form-data, and Browser-compatible headers. The 'Follow Redirects' checkbox is checked.

# Jmeter : tests fonctionnels et charges

## For Each et Loop Controller

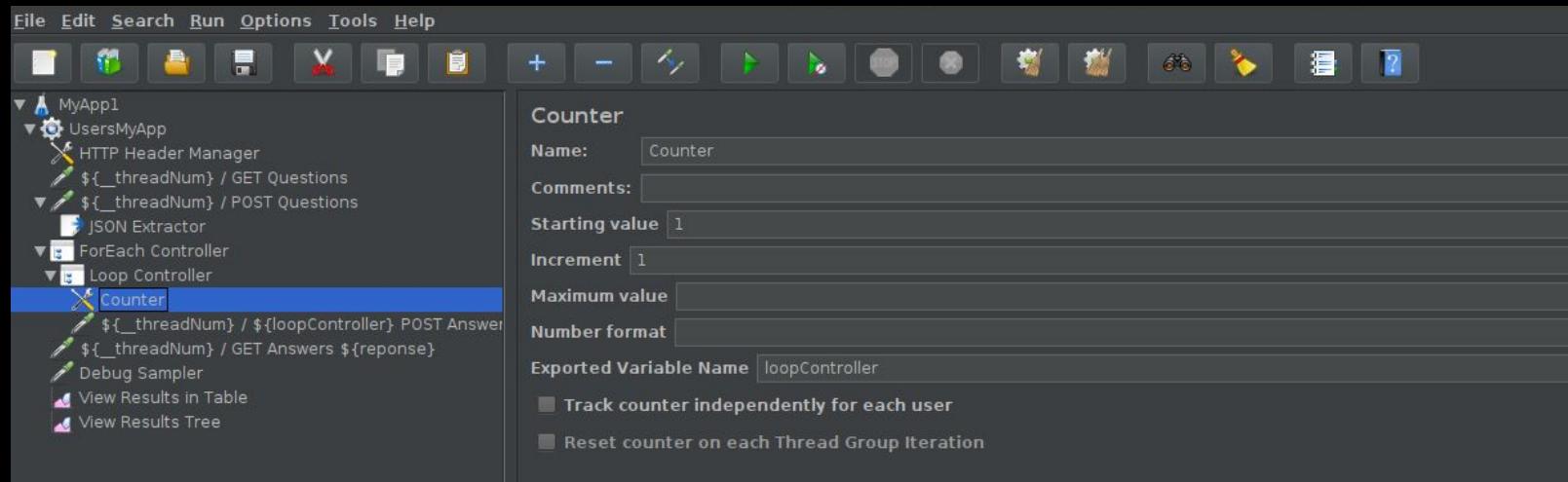
- plusieurs réponses



# Jmeter : tests fonctionnels et charges

## For Each et Loop Controller

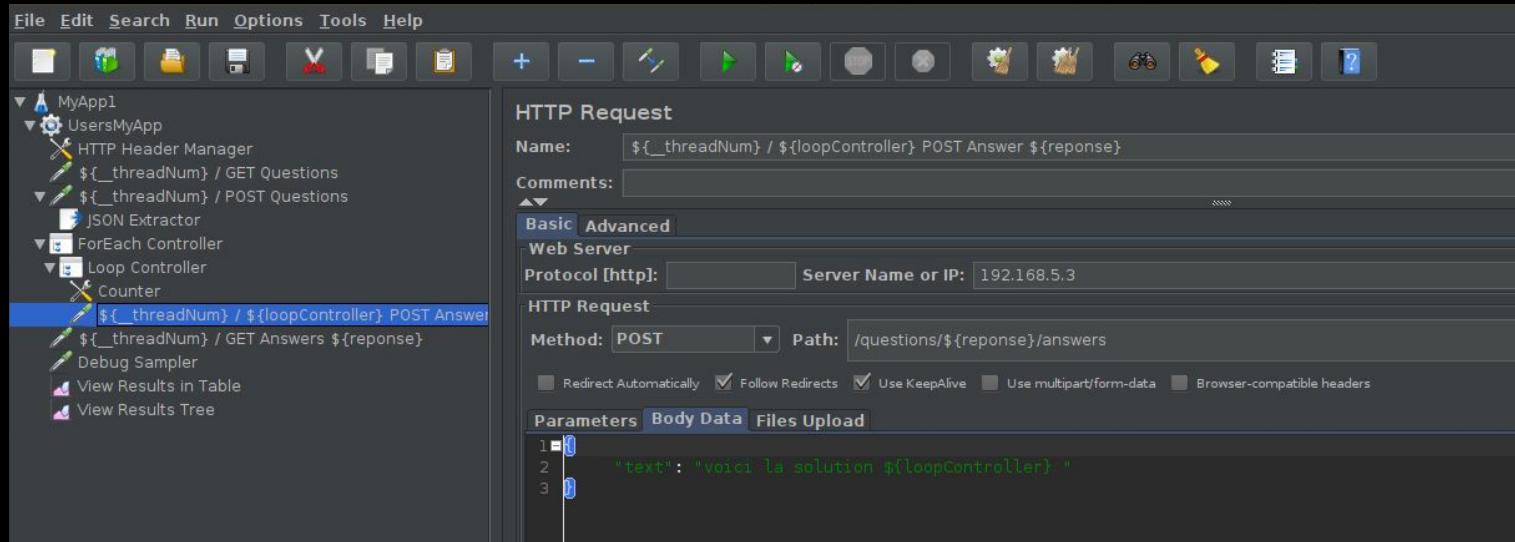
- plusieurs réponses



# Jmeter : tests fonctionnels et charges

## For Each et Loop Controller

- plusieurs réponses



# Jmeter : tests fonctionnels et charges

## Assertions - checks

- code retour http

The screenshot shows the JMeter interface with a focus on the 'Response Assertion' configuration dialog. The 'Test Plan' tree on the left includes a Thread Group, \${\_\_threadNum}/GET Questions, \${\_\_threadNum}/POST Questions (which contains a JSON Extractor and variables like codeTest, jsonTest-titre, jsonTest-description), ForEach Controller, Loop Controller (with a Counter and a POST sampler), and \${\_\_threadNum}/GET Response. The \${\_\_threadNum}/POST Questions node is selected.

**Response Assertion**

Name: `codeTest`

Comments:

Apply to:

Main sample and sub-samples  Main sample only  Sub-samples only  JMeter Variable Name to use

Field to Test

Text Response  Response Code  Response Message  Response Headers  
 Request Headers  URL Sampled  Document (text)  Ignore Status  
 Request Data

Pattern Matching Rules

Contains  Matches  Equals  Substring  Not  Or

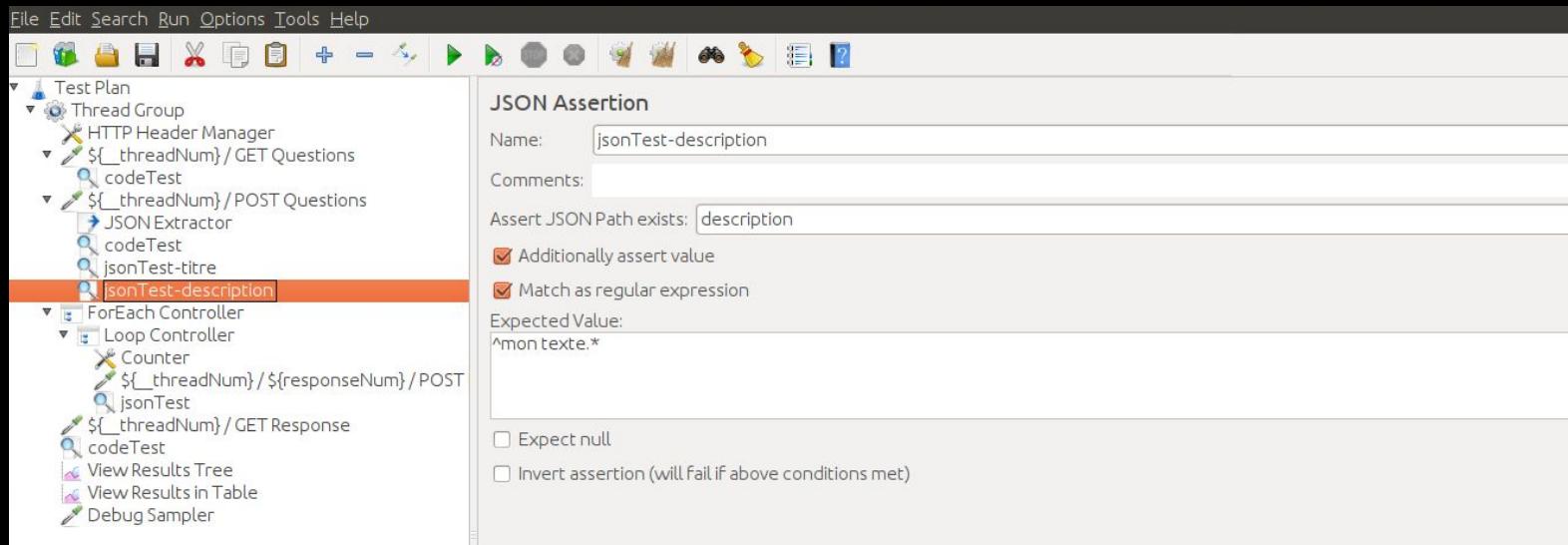
Patterns to Test

1 ^20.

# Jmeter : tests fonctionnels et charges

## Assertions - checks

- json response



# Jmeter : tests fonctionnels et charges

## Assertions - checks

- json response

The screenshot shows the JMeter interface with a focus on a 'JSON Assertion' configuration. The left pane displays a hierarchical 'Test Plan' structure with various samplers and controllers. The 'jsonTest' sampler, which is the current selection, is highlighted with an orange bar at the bottom of the tree. The right pane contains the configuration for this assertion:

- Name:** jsonTest
- Comments:** (empty)
- Assert JSON Path exists:** text
- Checkboxes:**  Additionally assert value,  Match as regular expression
- Expected Value:** `.*le retour ${responseNum}.*`
- Checkboxes (disabled):**  Expect null,  Invert assertion (will fail if above conditions met)

# Jmeter : tests fonctionnels et charges

## JSR223 Sampler - Variables

- centraliser la définition des variables... et bien plus

The screenshot shows the JMeter Test Plan interface. On the left, the tree view displays the test plan structure:

- Test Plan
- Thread Group
  - JSR223 Sampler (highlighted)
  - HTTP Header Manager
  - \${\_\_threadNum} / GET Questions
    - testCode
  - \${\_\_threadNum} / POST Questions
    - testCode
    - testJson - description
    - testJson - title
    - JSON Extractor
  - ForEach Controller
    - Loop Controller
      - Counter
      - \${\_\_threadNum} / \${responseNum} / POST
        - testCode
        - testJson - content
  - \${\_\_threadNum} / GET Response
    - testJson
    - testCode
    - codeTest
  - View Results Tree
  - View Results in Table
  - Debug Sampler

### JSR223 Sampler

Name: JSR223 Sampler

Comments:

Script language (e.g. groovy, beanshell, javascript, jexl...)

Language: groovy (Groovy 2.4.16 / Groovy Scripting Engine 2.0)

Parameters passed to script (exposed as 'Parameters' (type String) and 'args' (type String[]))

Parameters:

Script file (overrides script)

File Name:

Script compilation caching

Cache compiled script if available:

Script (variables: ctx vars props SampleResult sampler log Label Filename Parameters args OUT)

Script:

```
1 def rand = ${__Random(0, 2)};
2 def titre = "Titre de mon Thread"
3 def question = []
4 def reponse = "voici ma solution"
5
6 question.add("qui es tu ?")
7 question.add("que fais tu ?")
8 question.add("que dire ?")
9
10 vars.put("titre", titre);
11 vars.put("question", question[rand]);
12 vars.put("reponse", reponse);
```

# Jmeter : tests fonctionnels et charges

## JSR223 Sampler - Variables

- centraliser la définition des variables... et bien plus

The screenshot shows the JMeter interface with a test plan containing several samplers and controllers. A specific JSR223 Sampler is highlighted with a red selection bar.

**Test Plan Structure:**

- Thread Group
  - JSR223 Sampler (highlighted)
  - HTTP Header Manager
  - HTTP Request Manager
  - \${\_\_threadNum} / GET Questions
    - testCode
  - \${\_\_threadNum} / POST Questions
    - testCode
    - testJson - description
    - testJson - title
    - JSON Extractor
  - ForEach Controller
    - Loop Controller
      - Counter
  - \${\_\_threadNum} / \${\_\_responseNum} / POST
    - testCode
    - testJson - content
  - \${\_\_threadNum} / GET Response
    - testJson
    - testCode
    - codeTest
    - View Results Tree
    - View Results in Table
    - Debug Sampler

# Jmeter : tests fonctionnels et charges

## JSR223 Sampler - Variables

- centraliser la définition des variables... et bien plus

The screenshot shows the JMeter Test Plan editor interface. On the left, the Test Plan tree view displays a structure of samplers and controllers. A specific JSON Assertion node is selected, highlighted with an orange border. The main panel on the right contains the configuration for this assertion.

**JSON Assertion**

Name: testJson - content

Comments:

Assert JSON Path exists: text

Additionally assert value

Match as regular expression

Expected Value:  
^\${response} \${responseNum}.\*\$

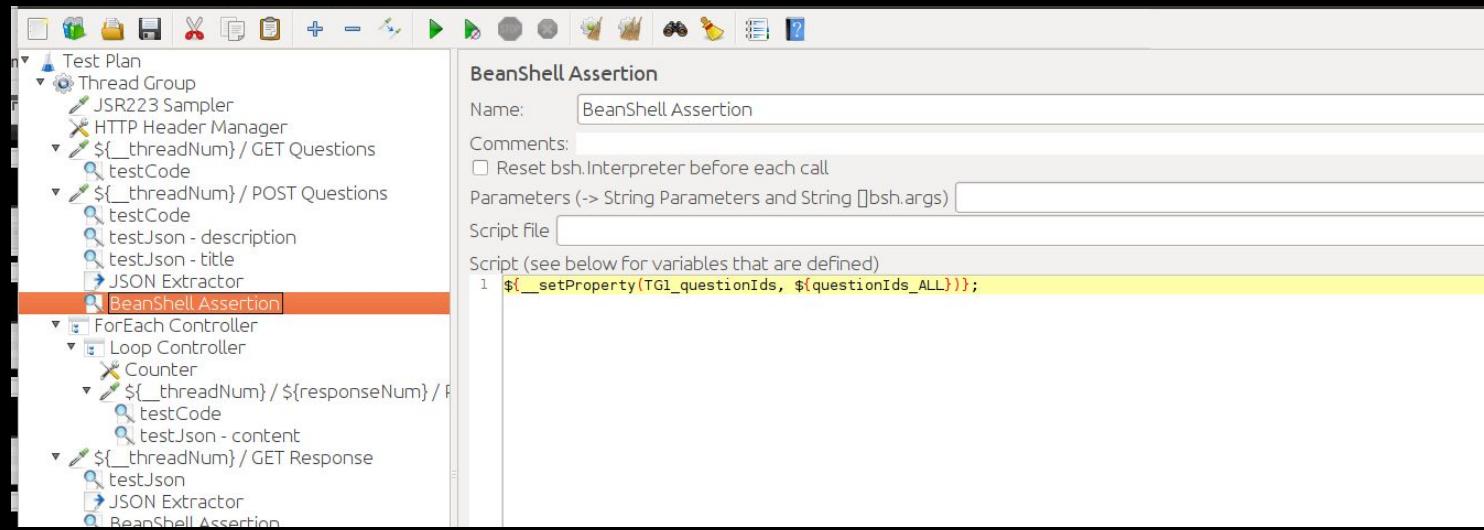
Expect null

Invert assertion (will fail if above conditions met)

# Jmeter : tests fonctionnels et charges

## JDBC

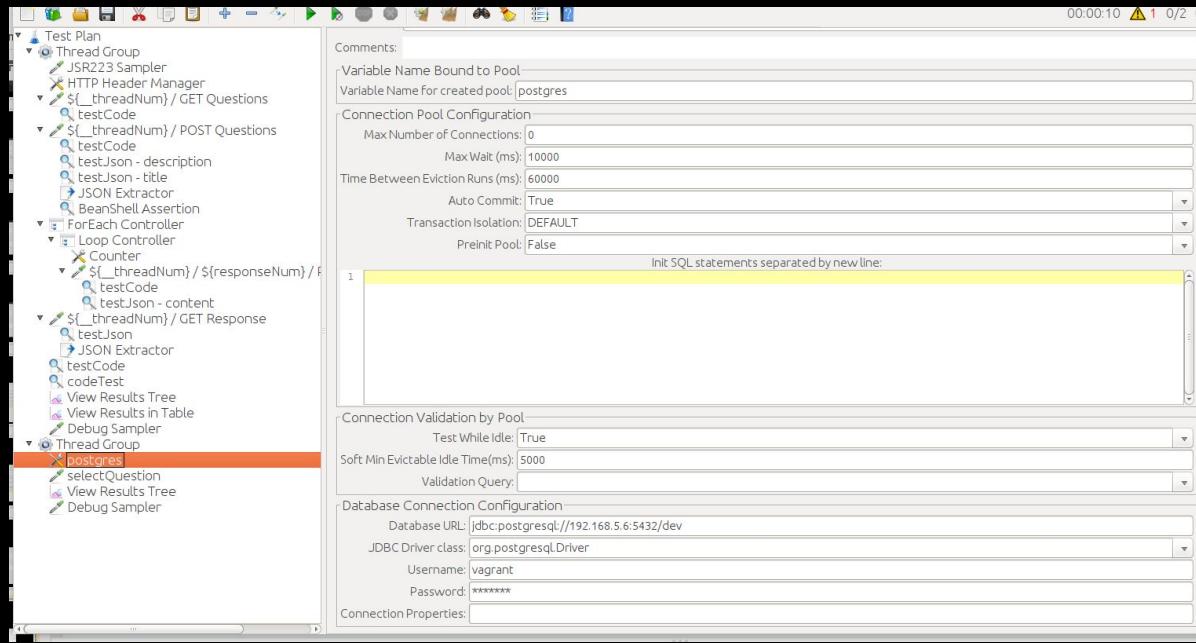
- tester votre base de données



# Jmeter : tests fonctionnels et charges

## JDBC

- tester votre base de données



# Jmeter : tests fonctionnels et charges

## JDBC

- tester votre base de données

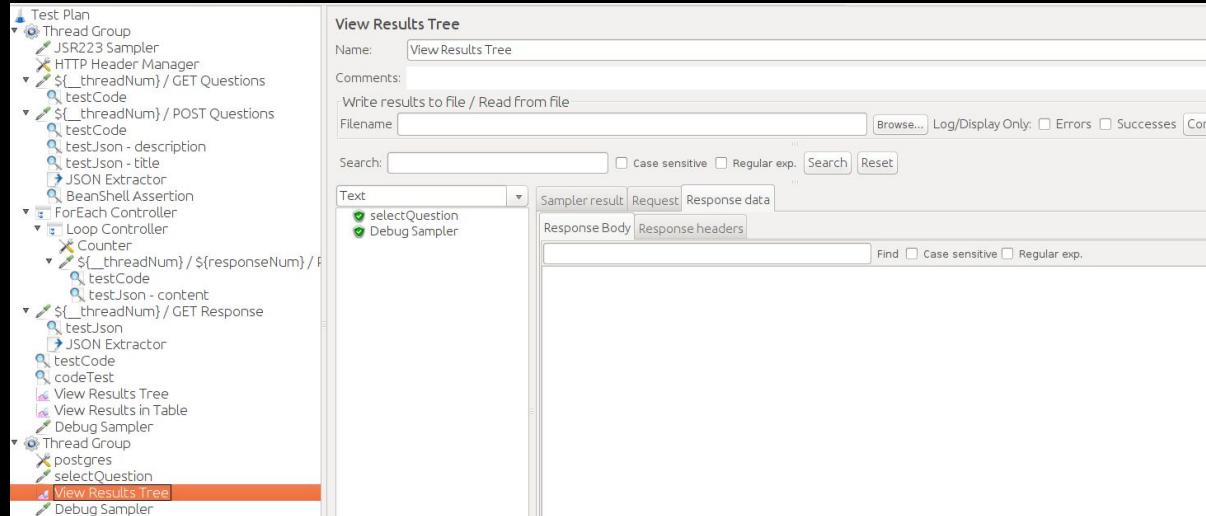
The screenshot shows the JMeter interface with a test plan containing the following structure:

- Test Plan
  - Thread Group
    - JSR223 Sampler
    - HTTP Header Manager
    - \${\_\_threadNum} / GET Questions
      - testCode
    - \${\_\_threadNum} / POST Questions
      - testCode
      - testJson - description
      - testJson - title
      - JSON Extractor
      - BeanShell Assertion
    - ForEach Controller
      - Loop Controller
        - Counter
        - \${\_\_threadNum} / \${responseNum} /
          - testCode
          - testJson - content
    - \${\_\_threadNum} / GET Response
      - testJson
      - JSON Extractor
      - testCode
      - codeTest
      - View Results Tree
      - View Results in Table
      - Debug Sampler
    - Thread Group
      - postgres
      - selectQuestion
      - View Results Tree
      - Debug Sampler

# Jmeter : tests fonctionnels et charges

## JDBC

- tester votre base de données : <https://jdbc.postgresql.org/download.html>



# Jmeter : tests fonctionnels et charges

## Intégration à Jenkins

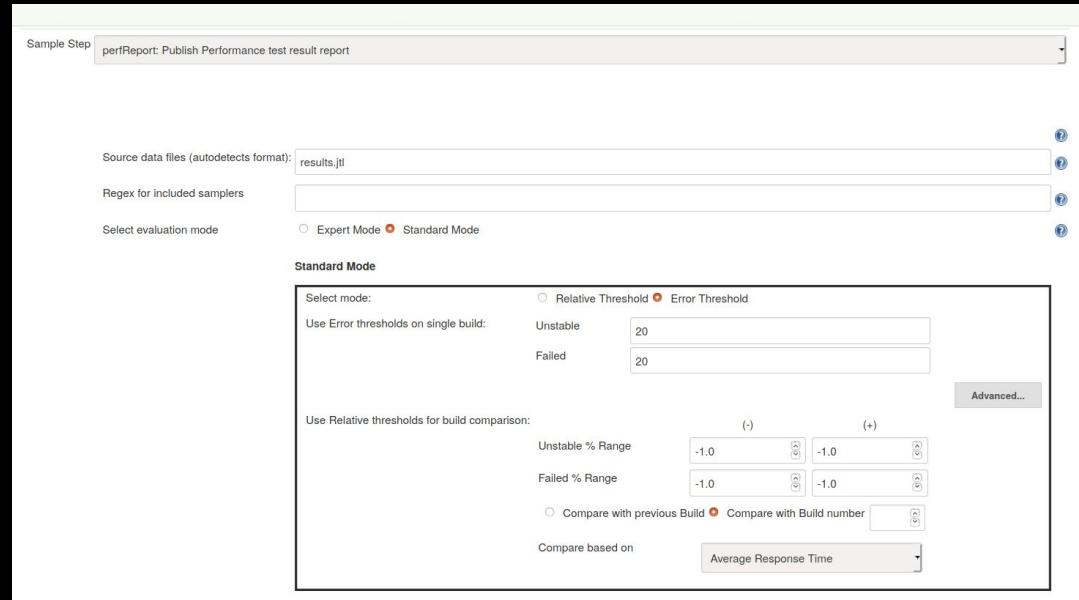
- ajustement du plan : ajout de critère de latence
- plugin “performance plugin”
- ajout de jmeter (cf vidéo précédente “premier run”)
- ajout au jenkinsfile (ex : dev)

```
if (branchName == "dev" ){  
  
    stage('JMETER - test'){  
        sh '/usr/bin/jmeter/apache-jmeter-5.2.1/bin/jmeter -n -t test/final_plan.jmx -l results.jtl'  
        sh 'cat results.jtl'  
        perfReport 'results.jtl'  
        /*perfReport errorFailedThreshold: 20, errorUnstableThreshold: 20, filterRegex: '', sourceDataFiles: 'results.jtl'*/  
    }  
}
```

# Jmeter : tests fonctionnels et charges

## Ajustement de l'interprétation résultats

- aide du pipeline syntax
- limite en % d'erreurs (choix...) : objectif fail le job



# Jmeter : tests fonctionnels et charges

## Ajustement de l'interprétation résultats

- aide du pipeline syntax
- limite en % d'erreurs (choix...) : objectif fail le job

```
if (branchName == "dev"){

    stage('JMETER - test'){
        sh '/usr/bin/jmeter/apache-jmeter-5.2.1/bin/jmeter -n -t test/final_plan.jmx -l results.jtl'
        sh 'cat results.jtl'
        perfReport errorFailedThreshold: 20, errorUnstableThreshold: 20, filterRegex: '', sourceDataFiles: 'results.jtl'
    }
}
```

# Jmeter : tests fonctionnels et charges

## Ajustement de l'interprétation résultats

- aide du pipeline syntax
- limite en % d'erreurs (choix...) : objectif fail le job

```
if (branchName == "dev"){

    stage('JMETER - test'){
        sh '/usr/bin/jmeter/apache-jmeter-5.2.1/bin/jmeter -n -t test/final_plan.jmx -l results.jtl'
        sh 'cat results.jtl'
        perfReport errorFailedThreshold: 20, errorUnstableThreshold: 20, filterRegex: '', sourceDataFiles: 'results.jtl'
    }
}
```

# Conclusions & Imperfections