

# JENKINS

## (l'essentiel)

### CI/CD

## Table des matières

<b>I - Jenkins (architecture, installation, ...)</b>	<b>3</b>
1. Premiers pas avec Hudson/Jenkins	3
1.1. Présentation et installation de Jenkins	3
1.2. Installation/démarrage de Jenkins sans tomcat	3
1.3. Configuration nécessaire lors du premier démarrage	4
1.4. Installation ou mise à jour de plugins pour Jenkins	4
1.5. Configuration élémentaire d'une tâche "jenkins / freeStyle"	4
1.6. job/item de type "pipeline"	6
2. Architecture de Jenkins (variantes)	7
2.1. Jenkins seul sans agent	7
2.2. Jenkins avec agent	8
2.3. Avec agents en mode "DinD" : Docker in Docker	8
2.4. Avec agents en mode "docker non imbriqué"	10
3. Jenkins avec docker	11
3.1. Installation de Jenkins via docker	11

3.2. Explication de la configuration (my-jenkins-config).....	13
3.3. Exemple de Pipeline pour node / javascript.....	15
3.4. Exemple de Pipeline pour maven / java.....	18
3.5. Exemple de Pipeline pour python.....	18

## II - Configuration de Jenkins.....20

1. Notifications des résultats (rss , email , ... ).....	20
1.1. Exemple de liens hypertextes sur résultats des builds.....	21
1.2. Notifications par flux rss.....	22
1.3. Notifications par email.....	23
2. Différents types de "builds" (avec Jenklins).....	25

## III - Annexe – Bibliographie, Liens WEB + TP.....29

1. Bibliographie et liens vers sites "internet".....	29
2. TP.....	29

# I - Jenkins (architecture, installation, ...)

## 1. Premiers pas avec Hudson/Jenkins

### 1.1. Présentation et installation de Jenkins

#### Premiers pas avec Jenkins

Jenkins est actuellement un **logiciel d'intégration continue** très en vogue car il est très simple à configurer et à utiliser.

#### Installation de Jenkins :

Recopier **jenkins.war** dans **TOMCAT\_HOME/webapps** (avec un éventuel Tomcat dédié à l'intégration continue configuré sur le port 8585 ou autre).

Etant donné que la configuration de Jenkins ne nécessite pas de base de données relationnelle (mais de simples fichiers sur le disque dur), il n'y a rien d'autre à configurer lors de l'installation.

Url de la console "jenkins" :

<http://localhost:8585/jenkins>

Premier menu à activer :

Administrer Jenkins / Configurer le système



### 1.2. Installation/démarrage de Jenkins sans tomcat

NB : il est également possible de démarrer une version récente de Jenkins sans serveur Tomcat via un script de ce genre :

**startJenkins.bat**

```
set JAVA_HOME=C:\Program Files\Java\jdk-17
set MVN_HOME=C:\Prog\apache-maven-3.8.4
set PATH="%JAVA_HOME%\bin";"%MVN_HOME%\bin";%PATH%
REM java -jar jenkins.jar -D"udson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true"
java -jar jenkins.war --httpPort=8585
```

et après un tel lancement l'URL menant à la console Jenkins sera simplement **<http://localhost:8585>**

## 1.3. Configuration nécessaire lors du premier démarrage

Lire le mot de passe temporaire à la console lors du premier démarrage (ex:  
`d1223a3ba2a44d079ecb7deec0625de8`)  
et reporter/recopier celui-ci dans la console de jenkins

Installer quelques plugins fondamentaux (ceux qui sont suggérés)

Configurer un compte principal (administrateur) pour les futurs démarrages :  
par exemple `username=admin password=admin123`

En configuration de TP, choisir l'URL `http://localhost:8585`

## 1.4. Installation ou mise à jour de plugins pour Jenkins

Menu "tableau de bord" / "Administrer Jenkins" / "Gestion des plugins"

## 1.5. Configuration élémentaire d'une tâche "jenkins / freeStyle"

Menu "tableau de bord" / "Nouveau item"

puis :

- donner un nom (ex : `jobXy`)
- choisir souvent "projet free-style" pour les cas simples/ordinaires
- OK

-----  
Dans la partie "gestion du code source", choisir généralement :

- GIT

et préciser l'url du référentiel git (par exemple <https://github.com/.../repoXy.git>)

*Attention: les versions récentes de Jenkins n'acceptent des URLS de type `file:///c:/xx/yy` qu'avec l'option `-D"hudson.plugins.git.GitSCM.ALLOW_LOCAL_CHECKOUT=true"` à fixer au démarrage*  
et dans la partie "branch to build" on pourra par exemple choisir `*/master` ou `*/main` .

-----  
Dans la partie "build" , choisir généralement :

- Invoquer les cibles Maven de haut niveau

et préciser la "cible (ou goal)" maven à déclencher (ex : `clean package`)

*NB: si le projet maven à construire est dans un sous répertoire du référentiel git (cas pas très conseillé mais admis), alors au niveau du build on peut préciser un chemin menant au pom.xml de type `sous_rep1/pom.xml` ou bien `sous_rep1/sous_sous_rep2/pom.xml` dans **config avancée** .*

Sauvegarder assez rapidement ces configurations essentielles.

Les configurations secondaires annexes pourront être ajoutées ultérieurement

## Lancement sur demande d'un "build" (associé à un job jenkins configuré)



## Affichage des résultats via la console de jenkins

La logique de navigation/sélection de jenkins est la suivante :

**Jenkins (server)** > **Job (name/type/config)** > **number of instance (with status/results)**

Exemple:

Jenkins > my-java-app1 > #4

Après avoir sélectionné un des niveaux , on accède à un menu (coté gauche) pour :

- \* créer/activer de nouveaux éléments
- \* (re)configurer plus en détails l'élément sélectionné
- \* afficher des détails sur l'élément sélectionné
- \* ...

Concernant les résultats d'un build, la partie la plus intéressante est souvent "*sortie console*" :



### Sortie de la console

```
-----  
[INFO] BUILD SUCCESS  
[INFO]  
-----  
[INFO] Total time: 24.003s  
[INFO] Finished at: Tue Apr 21 14:51:42 CEST 2015  
[INFO] Final Memory: 12M/32M  
[INFO]  
-----
```

## 1.6. job/item de type "pipeline"

Un job de type "pipeline" est assez conseillé au sein de jenkins car :

- il est grandement configurable/extensible
- il peut comporter plusieurs étapes (enchaînement en pipeline)
- un script de type "pipeline\_jenkins" peut être placé dans un référentiel git et ainsi être versionné

### Exemple de pipeline simple pour un projet java :

```

pipeline {
  agent any
  stages {
    stage('SCM') {
      steps {
        git url : 'https://github.com/didier-tp/test_junit.git' , branch : 'main'
      }
    }
    stage('Build') {
      steps {
        script {
          dir('with_mockito') {
            //sh "mvn -Dmaven.test.failure.ignore=true clean package"
            bat "mvn -Dmaven.test.failure.ignore=true clean package"
          }
        }
      }
      post {
        // If Maven was able to run the tests, even if some of the test failed, .....
        success {
          script {
            dir('with_mockito') {
              bat "mvn javadoc:javadoc"
              echo "javadoc generated , ..."
            }
          }
        }
      }
    }
    stage('sonar scan or prepa_docker') {
      steps {
        echo "sonar scan ou construction container docker (souvent sous linux)"
      }
    }
  }
}

```

NB: pas besoin de

```

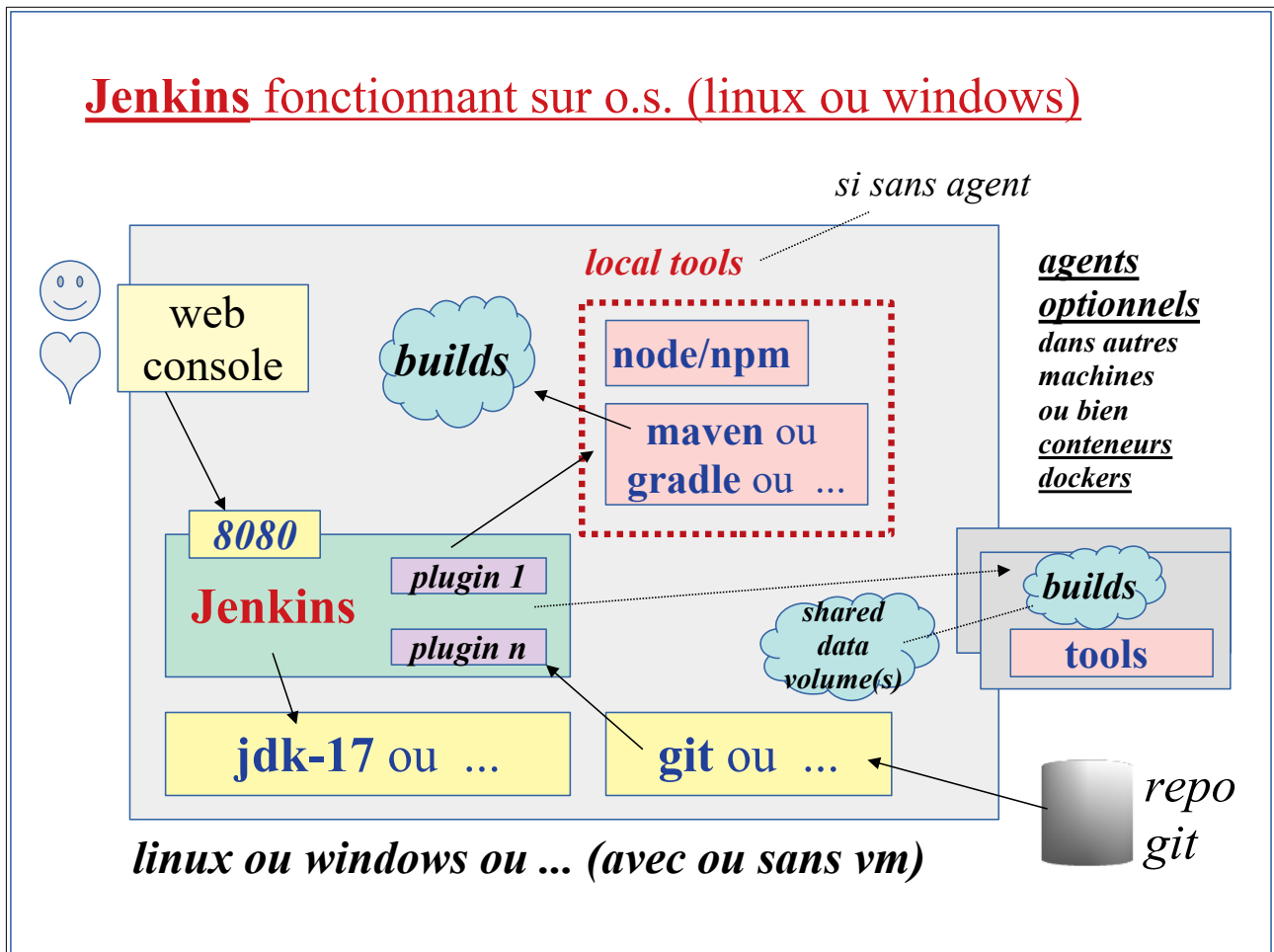
script {
  dir('with_mockito') {
  }
}

```

si pom.xml directement à la racine du référentiel git

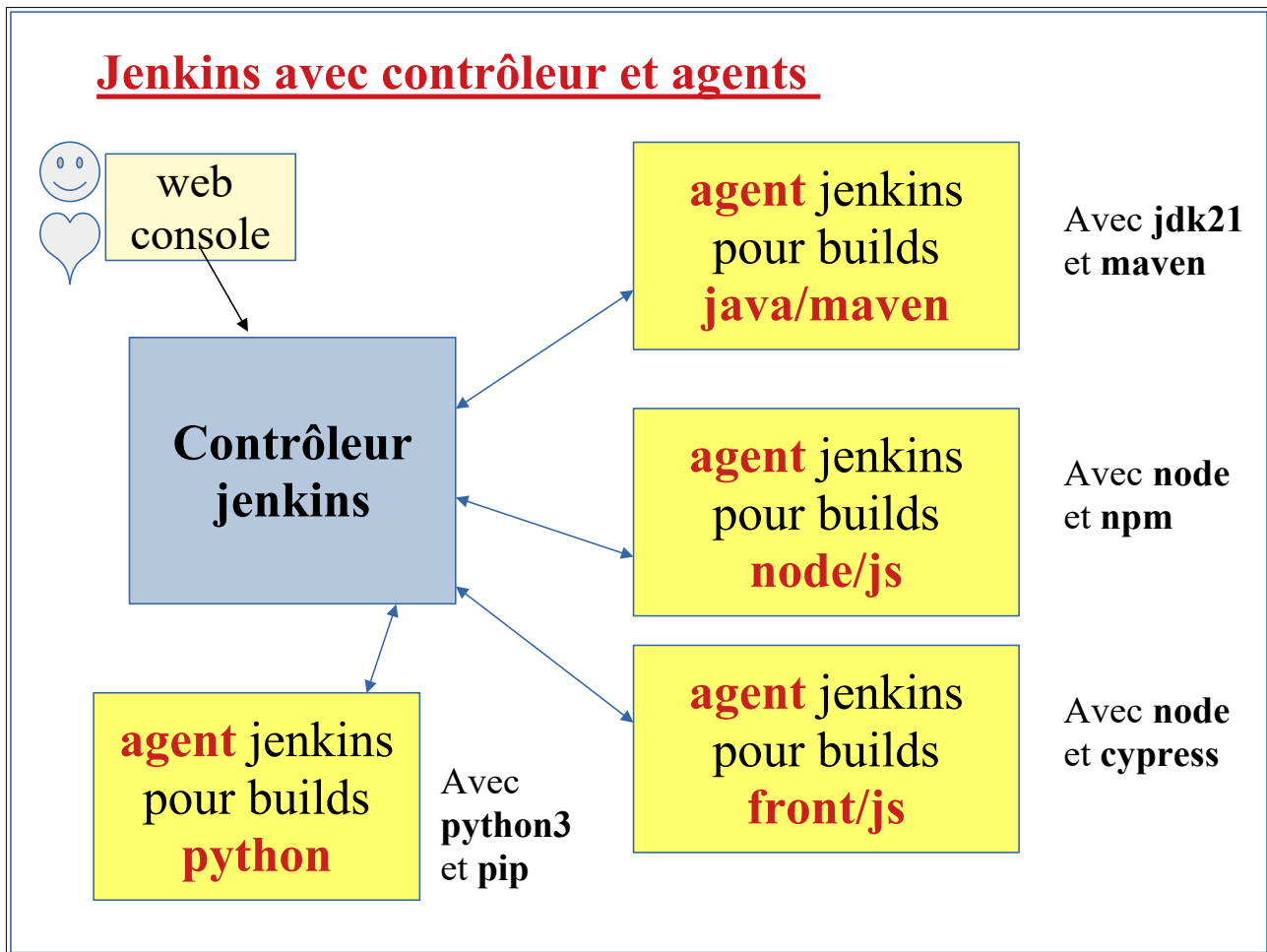
## 2. Architecture de Jenkins (variantes)

### 2.1. Jenkins seul sans agent



Dans ce mode (assez basique), le serveur Jenkins fonctionne en mode mono bloc et il doit être accompagné de nombreux outils (jdk, maven , node , npm , ...) si l'on souhaite pouvoir déclencher des "builds" avec plusieurs technologies (java, js, python, ...).

## 2.2. Jenkins avec agent



Dans ce mode plus sophistiqué (conseillé), la partie principale de jenkins va déléguer certains builds à des agents spécialisés dans certaines technologies :

- un agent sera spécialisé "builds java/maven" avec outils jdk et mven
- d'autres agents seront spécialisés "builds node/npm" ou "builds python" ou "build frontend" .

NB : La technologie "docker" peut éventuellement être utilisée pour simplifier la mise en place d'une telle configuration.

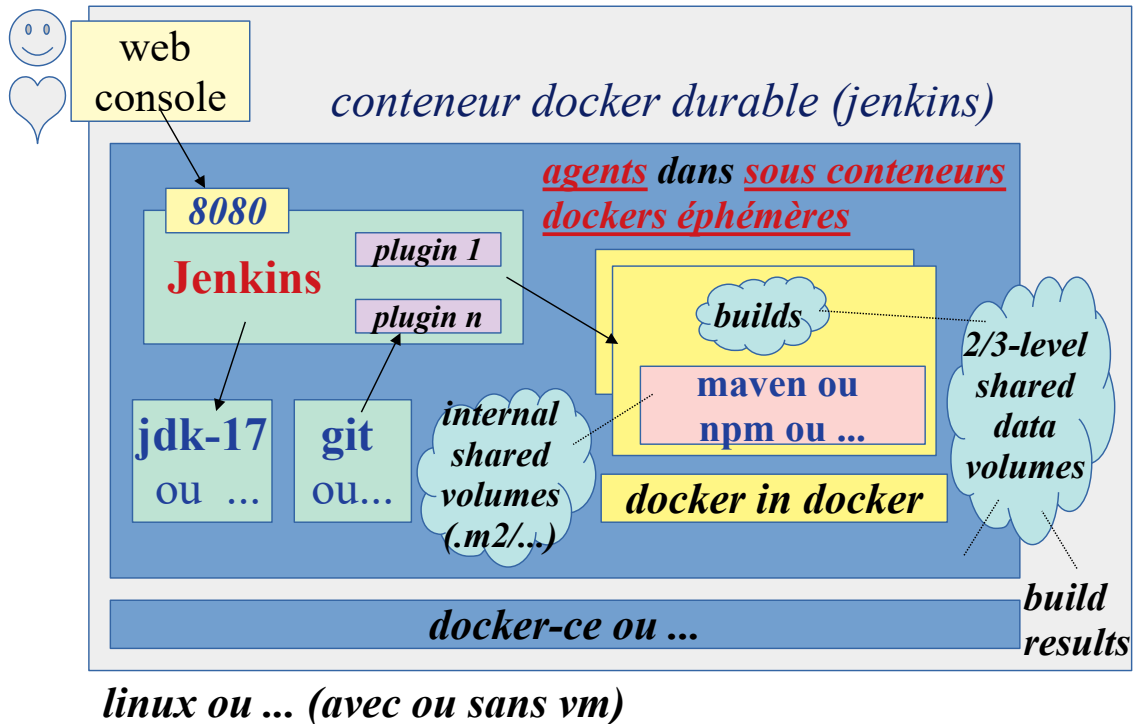
Le contrôleur jenkins et des agents peuvent fonctionner au sein de conteneurs "docker" que l'on peut gérer via "docker-compose" .

## 2.3. Avec agents en mode "DinD" : Docker in Docker

Si le contrôleur "Jenkins" fonctionne au sein d'un conteneur docker, on peut alors avoir des agents qui fonctionnent comme des conteneurs dockers imbriqués (en mode "DinD : docker in docker"). Cette solution est assez souple et flexible mais d'un point de vue "performance", c'est assez moyen.



## Jenkins avec agents en mode docker-in-docker



## Jenkins pipeline (build as code) with docker agent

```

pipeline {
  agent {
    docker {
      image 'maven:3-alpine'
      args '-v /root/.m2:/root/.m2'
    }
  }
  stages {
    stage('Build') {
      steps {
        git 'https://github.com/didier-mycontrib/env-ic-my-java-rest-app'
        sh "mvn -Dmaven.test.failure.ignore=true clean package"
      }
      post {
        success {
          echo 'with success'
        }
      }
    }
  }
}

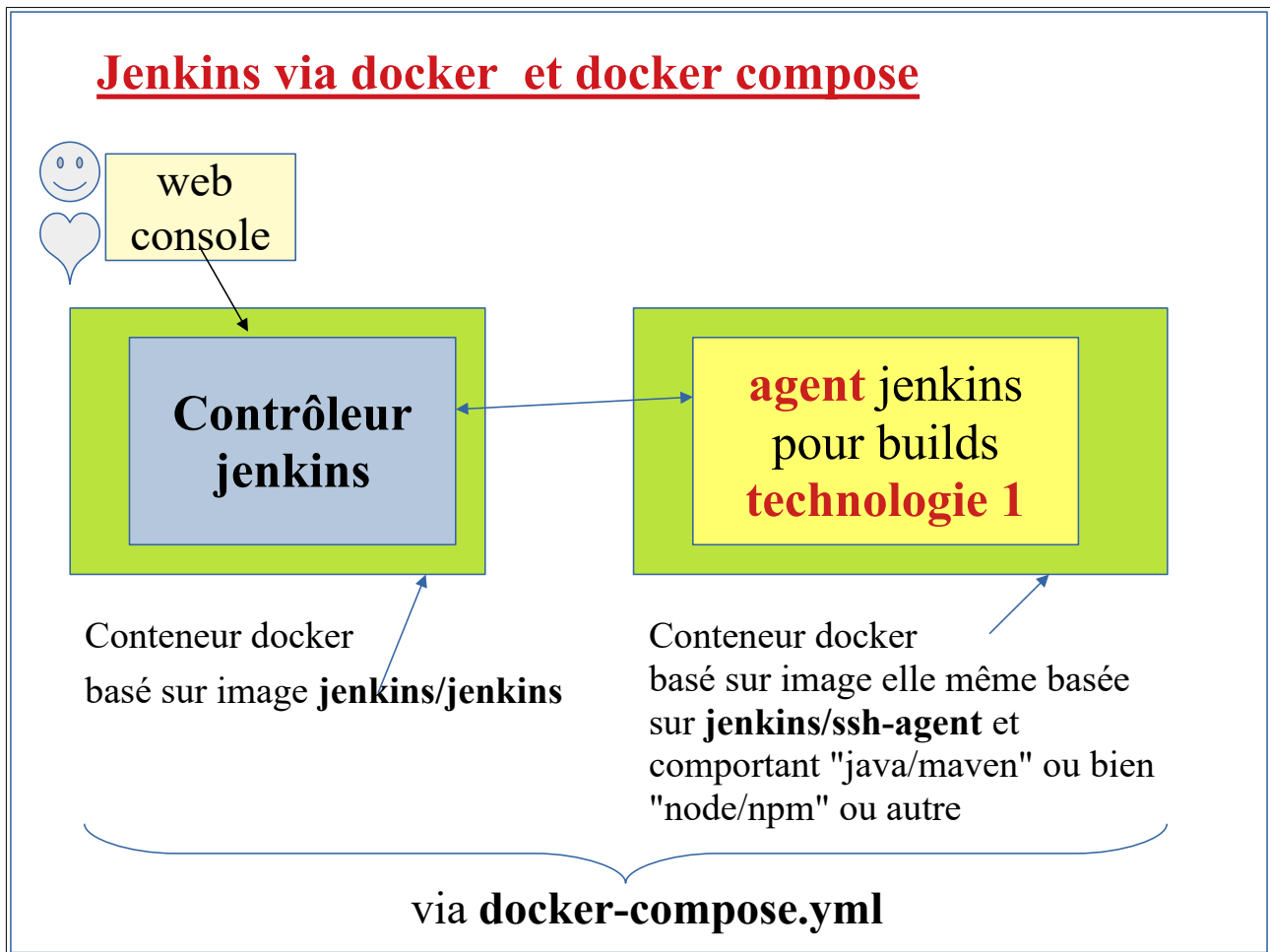
```

image (java ou ...) d'un conteneur docker éphémère (agent)

mapping de volume pour que le repo local .m2/... ne soit pas re-téléchargé à chaque lancement du build

succession d'étapes très flexible

## 2.4. Avec agents en mode "docker non imbriqué"



Ce mode de fonctionnement peut comporter lui-même plein de variantes :

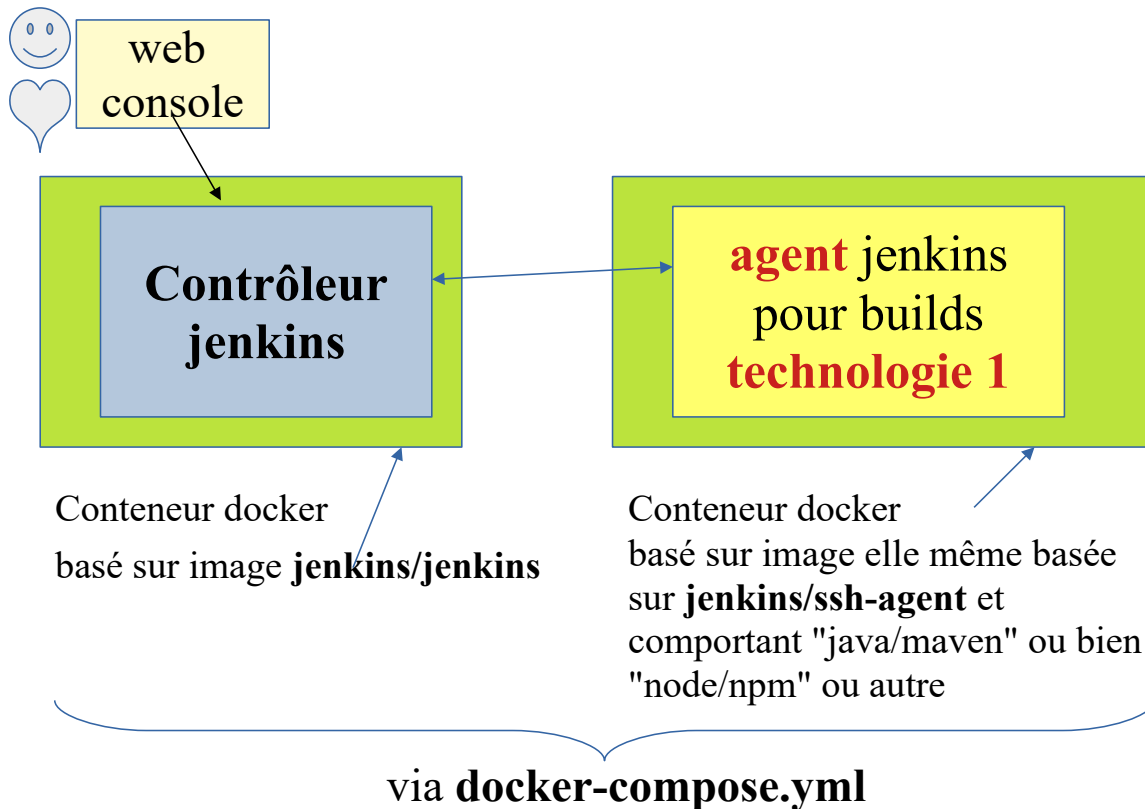
- conteneurs dockers fonctionnant (ou pas) sur des machines séparées
- en mode "static" ou "cloud élastique"
- ...

La technologie "docker compose" sera souvent la bienvenue pour configurer et démarrer un assemblage bien cohérent.

L'exemple officiel "<https://github.com/jenkins-docs/quickstart-tutorials>" montre une bonne base de configuration "jenkins/docker/docker-compose" pour un bon paquet de technologies (java/main, node/js, python, ...).

### 3. Jenkins avec docker

#### Jenkins via docker et docker compose



#### 3.1. Installation de Jenkins via docker

Sur une machine (éventuellement virtuelle) linux de type debian ou Ubuntu (fonctionnant par exemple avec VirtualBox && Vagrant ou WSL2) , on pourra enchaîner les installations suivantes:

- 1) installer si besoin GIT (apt install git , git config ...) et vérifier via git --version
- 2) se placer quelque part sur la machine virtuelle linux (ex: /home/xyz/local-git-repositories) puis cloner le référentiel git qui comporte une configuration "docker-jenkins" :  
**git clone https://github.com/didier-mycontrib/my-jenkins-config**
- 3) installer si besoin docker et docker-compose en lançant  
**sudo sh install-docker.sh** (ou un équivalent)  
où install-docker.sh est un script que l'on pourra trouver dans **my-jenkins-config/scripts/debian** ou bien **.../scripts/ubuntu**  
NB: selon OS exact (debian ou ubuntu) , il faudra peut être se déconnecter/reconnecter

pour que votre compte linux soit bien associé au groupe "docker" permettant de lancer docker sans sudo.

Vérifications: **docker ps**

**docker compose ps**

4) lancer l'installation de jenkins via docker compose en procédant de cette manière:

- se placer dans le répertoire .../local-git-repositories/**my-jenkins-config**
- vérifier que ce répertoire comporte bien le fichier **docker-compose.yaml** ainsi que le sous répertoire dockerfiles (contenant lui même plein de choses)
- lire de fichier README\_start\_stop.txt et lancer une des commandes suivantes:

**docker compose --profile python up -d**

**docker compose --profile node up -d**

**docker compose --profile maven up -d**

**docker compose --profile maven --profile node up -d**

...

- vérifier le bon démarrage de jenkins (partie contrôler + partie agent pour ...) via la commande **docker container ls**

5) lancer un navigateur pouvant accéder aux serveurs démarrés au sein de la VM linux

et depuis celui-ci spécifier l'URL suivante **http://localhost:8080**

s'identifier en tant qu'admin via **admin/admin**

6) utiliser la console de Jenkins pour configurer un item/job de type "pipeline"

en mode "pipeline script" ou bien "pipeline from scm":

URL GIT 1 (python) : [https://github.com/didier-tp/my\\_python\\_rest\\_api](https://github.com/didier-tp/my_python_rest_api) , branch : main

on pourra s'inspirer des exemples du répertoire .../my\_python\_rest\_api/jenkins/

URL GIT 2 (javascript): [https://github.com/didier-tp/vanilla\\_cypress\\_frontend](https://github.com/didier-tp/vanilla_cypress_frontend) , branch : main

URL GIT 3 (java): [https://github.com/didier-tp/spring6\\_2024](https://github.com/didier-tp/spring6_2024) , branch : main

NB:

après un redémarrage complet de la VM linux , on pourra éventuellement relancer **docker compose --profile ..... up -d** si docker ps montre qu'un agent n'a pas redémarré.

-----

Si besoin de tout arrêter du coté jenkins (sans suppression de la configuration) :

**docker rm -f desktop-jenkins\_agent-1-node**

**docker rm -f desktop-jenkins\_agent-1-maven**

**docker rm -f my-jenkins-config-jenkins\_controller-1**

**compose --profile python down**

-----

Si besoin de tout arrêter/ré-initialiser du coté jenkins (en supprimant configuration):

**docker compose --profile python down -v --remove-orphans**

-----

Si besoin de peaufiner un agent de Jenkins:

- arrêter les choses via **docker compose ... down ...**

- supprimer l'ancienne image de l'agent (via **docker image rm ....**)

- améliorer si besoin **dockerfiles/xyz/Dockerfile** et **docker-compose.yaml**

- relancer **docker compose --profile ..... up -d**

- vérifier le bon/meilleur fonctionnement au niveau des jobs de jenkins

## 3.2. Explication de la configuration (my-jenkins-config)

<https://github.com/didier-mycontrib/my-jenkins-config> est un fork de <https://github.com/jenkins-docs/quickstart-tutorials> avec les ajustements suivants :

La partie essentielle [docker-compose.yaml](#) a été adaptée (auto restart of agents , only python, maven, node).

La partie dockerfiles/**plugins.txt** a été améliorée en y ajoutant **docker-workflow:596.v3e6972b\_46b\_e2** sachant que le plugin "Docker pipeline" se télécharge via l'alias docker-workflow et par exemple en version 596.v3e6972b\_46b\_e2

Pour que l'on puisse fabriquer des images docker , les parties dockerfiles/python/Dockerfile , dockerfiles/maven/Dockerfile , dockerfiles/node/Dockerfile ont été améliorées de manière à ce que l'on puisse accéder à docker en mode ~~did~~ **dood** depuis l'utilisateur "jenkins" .

dockerfiles/node/Dockerfile (avec améliorations "docker" et "cypress" )

```
FROM jenkins/ssh-agent:6.9.0 as ssh-agent
ARG NODE_MAJOR=22

# ca-certificates because curl uses certificates from ca-certificates
RUN apt-get update && apt-get install -y --no-install-recommends ca-certificates curl gnupg && \
    # Installing nodejs
    mkdir -p /etc/apt/keyrings && \
    curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | gpg --dearmor -o \
/etc/apt/keyrings/nodesource.gpg && \
    echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] \
https://deb.nodesource.com/node_${NODE_MAJOR}.x nodistro main" | tee \
/etc/apt/sources.list.d/nodesource.list && \
    apt-get update && apt-get install nodejs -y && apt-get clean && rm -rf /var/lib/apt/lists/*

#NB: npm install -g ... not authorized for jenkins user
#but jenkins user can use (npm install -g ...) prepared by RUN ... as root
#so this image have to prepare npm install -g typescript , cypress , ....

# Set SHELL flags for RUN commands to allow -e and pipefail
# Rationale:https://github.com/hadolint/hadolint/wiki/DL4006
SHELL ["/bin/bash", "-eo", "pipefail", "-c"]

RUN echo "PATH=${PATH}" >> /etc/environment && chown -R jenkins:jenkins "${JENKINS_AGENT_HOME}"

#install docker-ce on this agent in order to build and push image from jenkins
RUN install -m 0755 -d /etc/apt/keyrings && \
    curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc \
&& \
    chmod a+r /etc/apt/keyrings/docker.asc && \
    echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
```

```
https://download.docker.com/linux/debian \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
tee /etc/apt/sources.list.d/docker.list > /dev/null &&\
apt-get update &&\
apt-get install -y --no-install-recommends acl docker-ce docker-ce-cli containerd.io docker-
buildx-plugin docker-compose-plugin

#gives jenkins user permissions to access /var/run/docker.sock
#via setfacl --modify user:jenkins:rw /var/run/docker.sock
#    npm config set prefix '/home/jenkins/.npm-global'
#    export PATH=/home/jenkins/.npm-global/bin:$PATH
#    in init_jenkins_access.sh
COPY init_jenkins_access.sh init_jenkins_access.sh

# override entrypoint of inherited jenkins/ssh-agent:6.9.0
# my_entrypoint.sh = call init_jenkins_access.sh and call setup-sshd
COPY my_entrypoint.sh my_entrypoint.sh
ENTRYPOINT ["sh", "my_entrypoint.sh"]

RUN npm install -g typescript && npm install -g http-server && npm install -g start-server-and-
test

#NB: xvfb and libglib-2.0.so is a required dependency of cypress
RUN apt-get update && \
    apt-get install --no-install-recommends -y \
    libgtk2.0-0 \
    libgtk-3-0 \
    libnotify-dev \
    libgconf-2-4 \
    libgbm-dev \
    libnss3 \
    libxss1 \
    libasound2 \
    libxtst6 \
    procps \
    xauth \
    xvfb

#switch to jenkins user for cypress install (.cache/Cypress) at goog place:
USER jenkins
RUN npx cypress install # Install Cypress binary into image

#switch to root user for good behavior of jenkins agent
USER root
```

init\_jenkins\_access.sh

```
#after apt-get install -y acl
setfacl --modify user:jenkins:rw /var/run/docker.sock
```

my\_entrypoint.sh

```
sh init_jenkins_access.sh
setup-sshd #entry_point of jenkins/ssh-agent:6.9.0
```

la partie **dockerfiles/jenkins.yaml** a été améliorée pour que l'on puisse éventuellement avoir plusieurs agents en parallèle avec des labels différents :

agent	labels
desktop-jenkins_agent-1-node	docker node node22 javascript without-maven without-python
desktop-jenkins_agent-1-maven	docker maven java jdk21 without-node without-python
desktop-jenkins_agent-1-python	docker python python3 without-maven without-node

Effets obtenus avec un démarrage via **--profile node --profile maven**

Tableau de bord / Administrer Jenkins/ Nodes (sur console <http://localhost:8080>)

Nodes							
<div> + New Node Configure Monitors </div>							
S	Nom	Architecture	Différence entre les horloges	Espace disque disponible	Espace de swap disponible	Free Temp Space	Temps de réponse
	contrôleur	Linux (amd64)	Synchronisé	216.22 GiB	2.00 GiB	216.22 GiB	0ms
	docker-ssh-jenkins-agent-maven	Linux (amd64)	Synchronisé	216.22 GiB	2.00 GiB	216.22 GiB	92ms
	docker-ssh-jenkins-agent-node	Linux (amd64)	Synchronisé	216.22 GiB	2.00 GiB	216.22 GiB	93ms
	docker-ssh-jenkins-agent-python		N/A	N/A	N/A	N/A	N/A

et

Labels					
node	node22	without-maven	without-python	javascript	docker

### 3.3. Exemple de Pipeline pour node / javascript

(ex : javascript , test cypress , backend node/express , frontend angular ou vue ou react , ...)

#### Jenkinsfile

```

pipeline {
    //agent any
    //agent { label 'node' }
    agent { label '! without-node' }
    environment{
        //NB: credential_dockerhub_didierdefrance69 is ID of credential
        //prepared in "Admin Jenkins / Credentials / system /global"
        dockerhub_credential_id='credential_dockerhub_didierdefrance69'
    }
}

```

```

        //dockerRegistry is dockerhub
        docker_registry= 'https://registry.hub.docker.com'

        docker_image_name='didierdefrance69/vanilla_cypress_frontend:1'
    }
    stages {
        //stage('from_git') {
        //    steps {
        //        git url : 'https://github.com/didier-tp/vanilla_cypress_frontend' , branch : 'main'
        //    }
        //}
        stage('npm_install') {
            steps {
                echo 'npm install'
                sh 'npm install'
            }
        }
        stage('tests') {
            steps {
                echo 'run ic script of package.json (start-server-and-test(http-server,3000,cypress))'
                sh 'npm run ic'
            }
        }
        stage('build_docker_image') {
            steps {
                //sh 'docker build -t didierdefrance69/vanilla_cypress_frontend:1 .'
                //with Pipeline docker plugin:
                script{
                    echo "docker_image_name=" + docker_image_name
                    dockerImage = docker.build(docker_image_name)
                }
            }
        }
        stage('push_docker_image') {
            steps {
                script{
                    echo "docker_registry=" + docker_registry
                    echo "dockerhub_credential_id=" +dockerhub_credential_id
                    //docker.withRegistry( docker_registry, dockerhub_credential_id ) {
                    //    dockerImage.push()
                    //    }
                }
            }
        }
    }
}

```

NB : la partie "push-docker-image" nécessite un compte sur dockerhub (ou bien un autre "docker-registry") et nécessite un paramétrage de **credential** au niveau de la console d'administration de Jenkins :



## New credentials

Type

Nom d'utilisateur et mot de passe

Portée ?

Global (Jenkins, agents, items, etc...)

Nom d'utilisateur ?

didierdefrance69

☐ Treat username as secret ?

Mot de passe ?

••••••••

ID ?

credential\_dockerhub\_didierdefrance69

Description ?

credential dont l'id doit correspondre avec celui utilisé au sein du pipeline

Create

### 3.4. Exemple de Pipeline pour maven / java

(ex : backend springBoot , test JUnit , librairie java , ...)

#### Jenkinsfile

```

pipeline {
    //agent any
    //agent { label 'maven' }
    agent { label '! without-maven' }
    stages {
        //stage('from_git') {
        //    steps {
        //        git url : 'https://github.com/didier-tp/spring6_2024', branch : 'main'
        //    }
        //}
        stage('mvn_clean_package_skip_test') {
            steps {
                script {
                    dir('tp/appliSpringV3') {
                        echo 'mvn clean package '
                        sh 'mvn clean package -Dmaven.test.skip=true'
                    }
                }
            }
        }
        stage('mvn test') {
            steps {
                script {
                    dir('tp/appliSpringV3') {
                        echo 'mvn test'
                        sh 'mvn test'
                    }
                }
            }
        }
    }
}

```

NB : les parties `script { dir('tp/appliSpringV3') { ... } }` ne sont pas nécessaire si pom.xml est à la racine du référentiel git .

### 3.5. Exemple de Pipeline pour python

(ex : python3 , pytest , venv , ...)

#### Jenkinsfile

```

pipeline {
    //agent any
    //agent { label 'python' }
    agent { label '! without-python' }
    stages {
        //stage('from_git') {
        //    steps {
        //        git url : 'https://github.com/didier-tp/my_python_rest_api', branch : 'main'
        //    }
        //}
        stage('build') {
            steps {
                sh 'python -m venv .venv && . .venv/bin/activate && python -m pip install -r requirements.txt'
                echo 'build'
            }
        }
        stage('tests') {

```

```
steps {
    sh '.venv/bin/activate && pytest --junit-xml test-reports/results.xml test_devise_api.py'
}
post {
    always {
        junit 'test-reports/results.xml'
    }
}
}
```

## II - Configuration de Jenkins

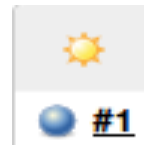
### 1. Notifications des résultats (rss , email , ...)

#### Notifier les développeurs du résultats des tests (état du projet)

Au niveau d'un logiciel d'intégration continue, on peut souvent paramétrer un mécanisme qui servira à avertir les développeurs sur :

- \* **résultats des tests** (statistiques , erreurs , ...)
- \* **état du projet (vert , rouge , ...)** et évolution
- \* ...

On parle souvent en terme imagé de **météo du projet** .



Les notifications peuvent être effectuées par :

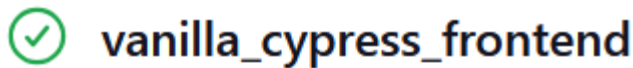
- \* des **envois d'e-mail** (à partir d' une liste des adresses e-mail des développeurs de l'équipe)
- \* des "**flux RSS**"
- \* une **simple consultation régulière du statut du projet dans jenkins**

**NB :** d'une version à l'autre de Jenkins , certaines présentations/informations peuvent changer.

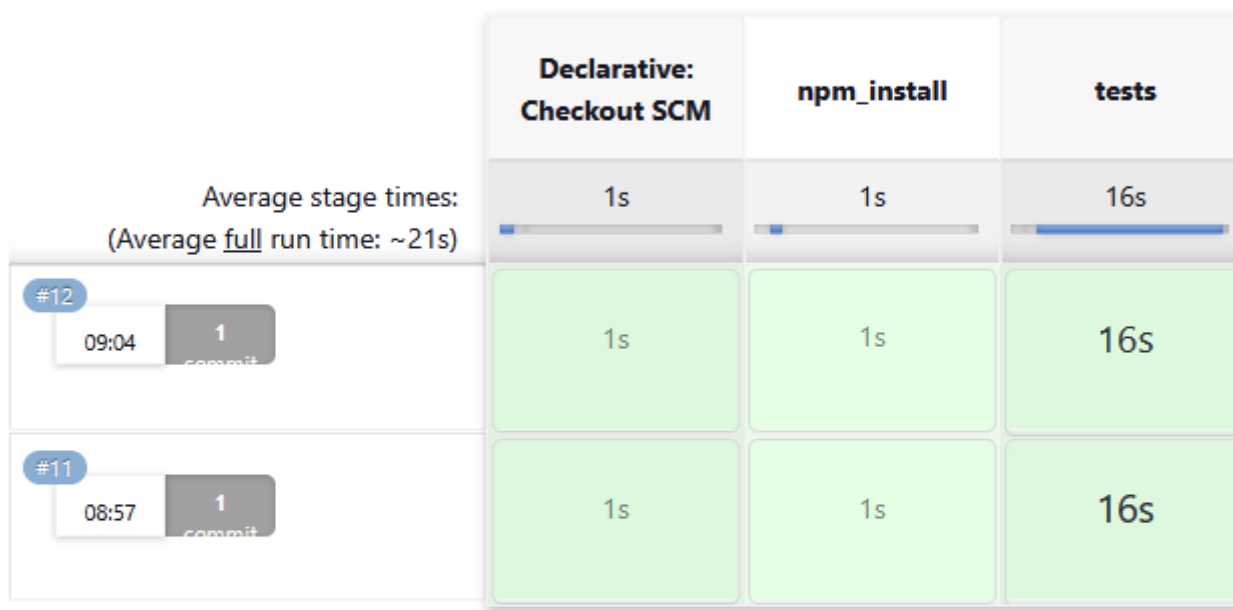
## 1.1. Exemple de liens hypertextes sur résultats des builds

Pour une consultation libre depuis un développeur du projet

exemple (ici sur job "vanilla\_cypress\_frontend" )



### Stage View



### Liens permanents

- [Dernier build \(#12\)](#), il y a 2 h 49 mn
- [Dernier build stable \(#12\)](#), il y a 2 h 49 mn
- [Dernier build avec succès \(#12\)](#), il y a 2 h 49 mn
- [Dernier build en échec \(#9\)](#), il y a 3 h 0 mn
- [Dernier build non réussi \(#9\)](#), il y a 3 h 0 mn
- [Dernier build complété \(#12\)](#), il y a 2 h 49 mn

[http://localhost:8080/job/vanilla\\_cypress\\_frontend/lastBuild/](http://localhost:8080/job/vanilla_cypress_frontend/lastBuild/)  
[http://localhost:8080/job/vanilla\\_cypress\\_frontend/lastStableBuild/](http://localhost:8080/job/vanilla_cypress_frontend/lastStableBuild/)  
[http://localhost:8080/job/vanilla\\_cypress\\_frontend/lastSuccessfulBuild/](http://localhost:8080/job/vanilla_cypress_frontend/lastSuccessfulBuild/)  
[http://localhost:8080/job/vanilla\\_cypress\\_frontend/lastFailedBuild/](http://localhost:8080/job/vanilla_cypress_frontend/lastFailedBuild/)

## 1.2. Notifications par flux rss

### Notifications élémentaires par flux rss (jenkins)



Jenkins créer des "flux RSS" attachés à chaque projet pris en charge.

Un flux RSS "tous les builds" permet d'être averti du résultat de chaque nouveau build .

Un flux RSS "tous les échecs" permet de n'être averti qu'en cas d'échec lors d'un build.

En cliquant sur l'un des icônes "RSS" de l'interface graphique de Jenkins , on peut (via le menu contextuel "copier l'adresse du lien") récupérer l'URL du flux (exemple : <http://localhost:8585/jenkins/job/my-java-app1/rssAll> ) pour ensuite paramétrer un lien dans un navigateur internet (tel que firefox) ou bien dans un logiciel de consultation des emails (tel que ThunderBird ou Outlook).

Pour accrocher un flux rss à *thunderbird*, le mode opératoire est le suivant :

.... / paramètres des comptes / gestions des comptes / nouveau compte de type "blog & news" / gérer les abonnements puis saisir l'adresse du flux et ajouter .

NB : le contenu du flux RSS comporte simplement un message "build ... réussi ou en échec" et un lien hypertexte qui renvoie sur la partie "web" de jenkins qui détaille les résultats .

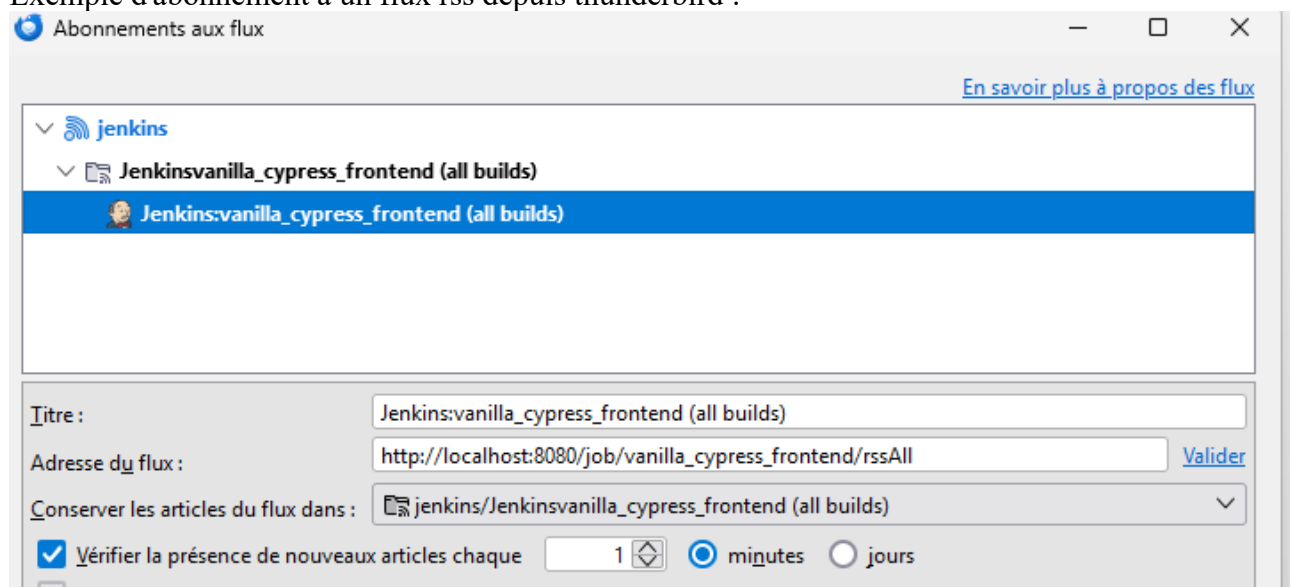
NB : au sein des versions récentes de jenkins , les flux rss sont moins mis en avant mais il sont encore disponibles (via ... / Atom feed / ...) :

[http://localhost:8080/job/vanilla\\_cypress\\_frontend/rssFailed](http://localhost:8080/job/vanilla_cypress_frontend/rssFailed)

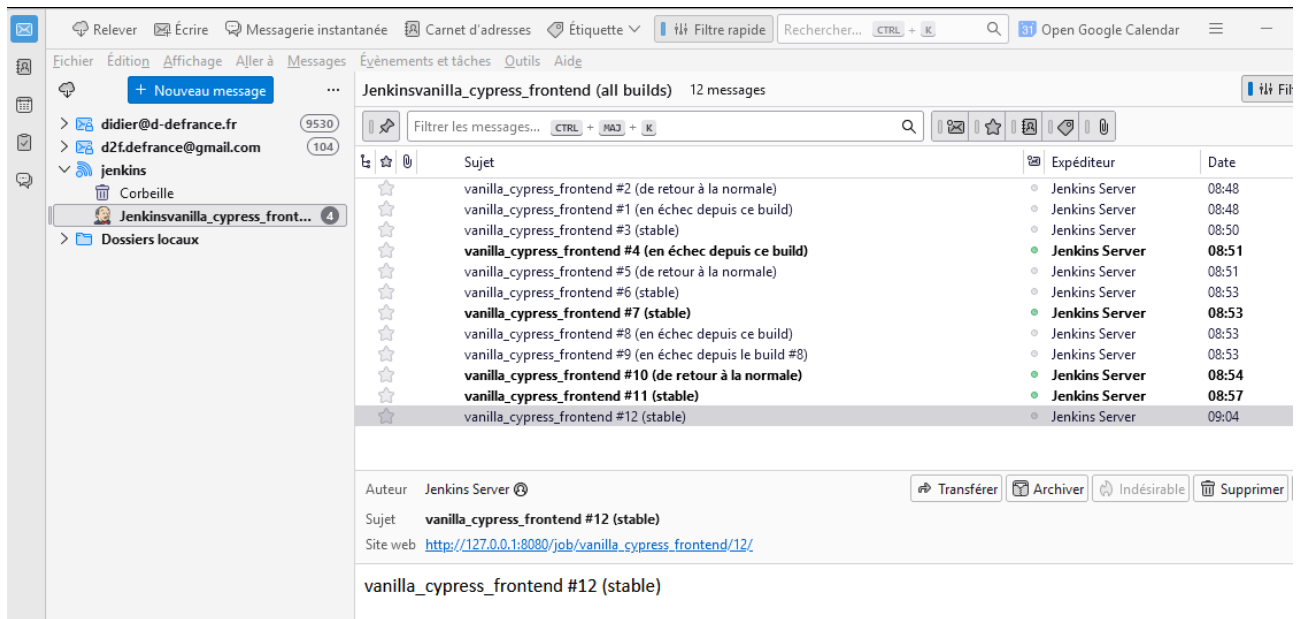
[http://localhost:8080/job/vanilla\\_cypress\\_frontend/rssAll](http://localhost:8080/job/vanilla_cypress_frontend/rssAll)

[http://localhost:8080/job/vanilla\\_cypress\\_frontend/rssLatest](http://localhost:8080/job/vanilla_cypress_frontend/rssLatest)

Exemple d'abonnement à un flux rss depuis thunderbird :



Exemple de notifications rss jenkins reçues au niveau de thunderbird :



Les messages rss reçus comportent un lien hypertexte vers une partie de la console "jenkins" de manière à obtenir plus de détails

### 1.3. Notifications par email

#### Notification par email (depuis Jenkins)

Pour activer la notification par emails , il faut avoir préalablement paramétrer un lien vers un serveur SMTP depuis la partie "administration" de la console de Jenkins.

Ensuite, au niveau d'un "**build**" (application à construire), on peut configurer près des "**post-action**" une **notification basique par email** (avertir seulement en cas d'échec) en **précisant une liste d'emails pour les développeurs destinataires** :

☒ Notification par email

Recipients

user1@localhost user2@localhost

☒ Send e-mail for every unstable build

*e-mail reçu en cas d'échec.*

Sujet **Jenkins build became unstable: my-java-app1 #3**  
Pour Moi ☆, user2@localhost ☆

See <http://localhost:8585/jenkins/job/my-java-app1/3/changes>

On peut éventuellement installer un plugin jenkins supplémentaire "**email-ext.hpi**" puis s'en servir via la post-action "**Editable Email Notification**" pour contrôler de façon plus fine les notifications envoyées.

Tableau de bord > Administrer Jenkins > System >

### Notification par email

Serveur SMTP

mail.gandi.net

Suffixe par défaut des emails des utilisateurs ?

Avancé ^ Edited

☒ Use SMTP Authentication ?


Nom d'utilisateur

didier@d-defrance.fr

Mot de passe

••••••

☒ Déverrouillez votre compte pour afficher les suggestions de saisie automatique

☒  Déverrouiller le compte

Port SMTP ?

465

Save Appliquer

Envoi d'un mail depuis une étape d'un pipeline :

```
stage('Send email') {
    def mailRecipients = "your_recipients@company.com"
    def jobName = currentBuild.fullDisplayName

    email body: "${SCRIPT, template='groovy-html.template'",
        mimeType: 'text/html',
        subject: "[Jenkins] ${jobName}",
        to: "${mailRecipients}",
        replyTo: "${mailRecipients}",
        recipientProviders: [[class: 'CulpritsRecipientProvider']]
}
```



## 2. Différents types de "builds" (avec Jenklins)

### Différents types de "builds"

Types de "build"	Commentaires/considérations
<b>Local / privé</b>	<b>Lancé manuellement</b> (et idéalement fréquemment) par le développeur (depuis son IDE) . → <i>Permet de savoir si ses propres changements fonctionnent</i>
<b>Intégration rapide (de jour)</b>	<b>Tests d'intégration rapides déclenchés après chaque commit ou bien régulièrement (ex : toutes les 20 minutes).</b> Seuls les tests rapides (unitaires + intégrations) sont lancés → <i>Permet de savoir si l'assemblage des changements de tous les développeur fonctionne .</i>
<b>Intégration journalière</b> poussée/sophistiquée à heure fixe <b>(nightly build)</b>	<b>Tests sophistiqués (longs)</b> , tests de performance , génération de documentation, de rapports , .... → <i>Permet de savoir si la dernière version produite du logiciel est en état de marche .</i>

## Réglages de fréquence via une syntaxe "crontab" (par exemple dans Jenkins) :

Syntaxe "crontab" :

**mm hh jj MM JJ [tâche]**

mm représente les minutes (de 0 à 59)

hh représente l'heure (de 0 à 23)

jj représente le numéro du jour du mois (de 1 à 31)

MM représente le numéro du mois (de 1 à 12)

JJ représente le numéro du jour dans la semaine  
(0 : dimanche , 1 : lundi , 6 : samedi , 7:dimanche)

Si, sur la même ligne, le « *numéro du jour du mois* » et le « *jour de la semaine* » sont renseignés, alors **cron** (ou ....) n'exécutera la *tâche* que quand ceux-ci coïncident .

.../...

## Réglage de la fréquence des "builds"

Pour chaque valeur numérique (mm, hh, jj, MMM, JJJ) les notations possibles sont :

\* : à chaque unité (0, 1, 2, 3, 4...)

5,8 : les unités 5 et 8

2-5 : les unités de 2 à 5 (2, 3, 4, 5)

\*/3 : toutes les 3 unités (0, 3, 6, 9...)

10-20/3 : toutes les 3 unités, entre la dixième et la vingtième  
(10, 13, 16, 19)

Et donc pour un nightly build d'intégration continue :

---> **0 3 \* \* 1-6** (*tous les jours à 3h du matin  
sauf les dimanches*)

et pour un build rapide de jour :

→ **\*/15 8-20 \* \* 1-5** (*tous les jours sauf les week-ends ,  
toutes les 15 minutes de 8h à 20h*)

## Lancement périodique (ex journalier) au niveau de Jenkins

☒ Construire périodiquement

Planning

0 3 \* \* 1-6

Ce champ suit la syntaxe de cron (avec des différences mineures). Chaque ligne consiste en 5 champs séparés par des TABs ou des espaces :

MINUTES HEURES JOURMOIS MOIS JOURSEMAINE

MINUTES Les minutes dans une heure (0-59)

HEURES Les heures dans une journée (0-23)

JOURMOIS Le jour dans un mois (1-31)

MOIS Le mois (1-12)

JOURSEMAINE Le jour de la semaine (0-7) où 0 et 7 représentent le dimanche

dans cet exemple : tous les jours (de lundi à samedi) à 3h du matin. "nightly build"

## Lancement sur détection périodique des changements (SVN/GIT)

☒ Scrutation de l'outil de gestion de version

Planning

\* /15 8-20 \* \* 1-5

dans cet exemple : Jenkins vérifie toutes les 15 minutes si certains changements ont eu lieu au niveau du référentiel de code source (de lundi à vendredi et de 8h à 20h). En cas de changement détecté → lancement (potentiellement très fréquent) d'un build d'intégration.

## Suppression des anciens builds (jenkins)

☒ Supprimer les anciens builds

Nombre de jours de conservation des builds

15

si non vide, les enregistrements de build seront conservés au maximum ce nombre de jours

Nombre maximum de builds à conserver

150

si non vide, pas plus de ce nombre de builds ne sera conservé

# ANNEXES

## III - Annexe – Bibliographie, Liens WEB + TP

### 1. Bibliographie et liens vers sites "internet"


### 2. TP