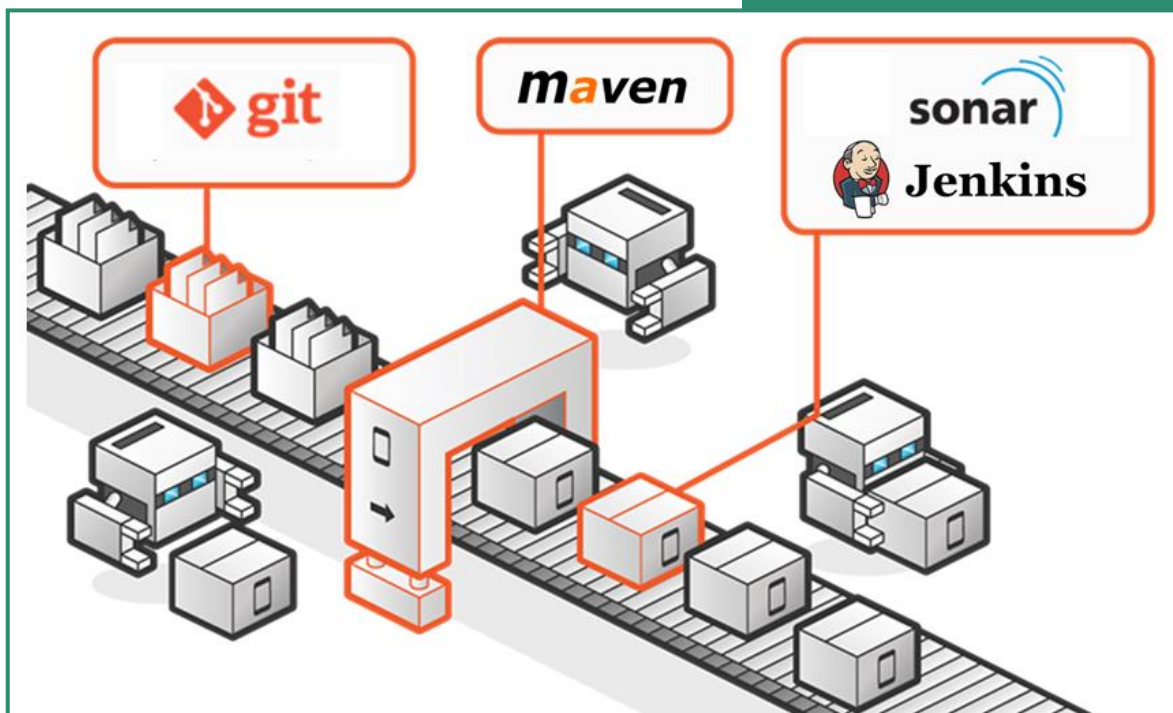


# Guide d'installation et d'utilisation d'une usine logicielle pour Java



**Promotion :** E5FI – 3<sup>ème</sup> année  
ingénieur en apprentissage – Filière  
Informatique

**Unité :** SI5I21

# Sommaire

Introduction .....	3
I. Vagrant .....	4
II. Ansible .....	4
III. Installation et utilisation de l'usine logicielle .....	4
1. Installation des outils .....	4
2. Installation de la VM et déploiement .....	5
3. Utilisation .....	5
4. Gestion de la machine virtuelle.....	7
IV. Détail du fichier Vagrantfile .....	8
V. Détail des scripts Ansible .....	9
1. Script principal : install_all.yml.....	9
2. Mise à jour du cache aptitude avec proxy.....	10
3. Git.....	10
4. Java .....	11
5. Apache .....	11
6. MySQL .....	13
7. Tomcat.....	14
8. Maven.....	14
9. Jenkins .....	15
VI. Détail de l'appli .....	18
Lexique.....	19
Webographie .....	20
Conclusion.....	21

# Introduction

---

Ce projet consiste à créer une usine logicielle J2EE composée de plusieurs logiciels permettant à une équipe de développeurs de travailler sur un projet en déploiement continu. Les différents composants permettent la centralisation du code écrit par l'équipe (repository git via Github), l'intégration continue (Jenkins), un gestionnaire de compilation (Maven), un serveur applicatif (Tomcat), un gestionnaire de repository (Nexus) ainsi qu'un auditeur de code (SonarQube). Ces deux derniers composants n'ont pu être traités dans le temps imparti.

De plus, ce projet propose le déploiement de cette usine logicielle de façon rapide et simple. Pour cela, notre rendu pour ce projet prend la forme d'une machine virtuelle installable via le logiciel Vagrant, d'une application J2EE et d'un ensemble de scripts Ansible permettant l'automatisation de l'installation et de la configuration des composants cités précédemment sur la machine virtuelle. Ainsi, un développeur intéressé par cette usine logicielle pourra la déployer en quelques commandes sur sa machine ou celles de son équipe.

Ce rapport présentera de façon succincte les outils Vagrant et Ansible. Puis, l'installation de l'usine logicielle sera expliquée ainsi que tous les détails concernant les processus d'installation et de configuration des composants de l'usine.

Toutes nos sources sont disponibles sur Github :

- Scripts Ansible et configuration de Vagrant : <https://github.com/harfangeek/ansible-scripts>
- Serveur de l'application : <https://github.com/harfangeek/api-tasks-java-ee>
- Client de l'application : <https://github.com/harfangeek/tasks-java-ee>

## I. Vagrant

Vagrant est un logiciel libre et open source pour la création et la configuration des environnements de développement virtuel. Il permet un déploiement rapide et facile de machine virtuelle. Vagrant fournit des images de systèmes d'exploitations préinstallées permettant ainsi une création rapide d'une nouvelle machine virtuelle. L'outil peut s'interfacer avec VirtualBox et VmWare ainsi qu'avec divers langages de déploiement par scripts tel que Ansible, Chef ou Shell.

## II. Ansible

Ansible est un outil permettant l'installation et la configuration automatisée d'un environnement à l'aide de scripts. Il permet l'installation, la configuration et le déploiement d'applications sur une ou plusieurs machines. Il se connecte via SSH aux machines cibles et ne nécessite pas d'installation sur celles-ci. Un script Ansible est écrit en YAML et se compose principalement d'une suite de tâche à effectuer (installation d'un paquet, écriture d'un fichier etc).

## III. Installation et utilisation de l'usine logicielle

### 1. Installation des outils

Pour déployer l'usine logicielle, nous avons besoin des outils suivant : Vagrant, VirtualBox, Ansible, et Git (pour la récupération des scripts). Pour la suite de ce rapport, nous considérons le déploiement de l'usine logicielle depuis une distribution Ubuntu 14.04. Le mode opératoire suivi est :

- Installation de vagrant (version 1.7.2) via un paquet debian
- Installation de Virtualbox, Ansible et Git via les dépôts Ubuntu :
  - `wget https://releases.hashicorp.com/vagrant/1.7.2/vagrant\_1.7.2\_x86\_64.deb`
  - `dpkg -i vagrant_1.7.2_x86_64.deb`
  - `apt-get install virtualbox virtualbox-dkms ansible git`

## 2. Installation de la VM et déploiement

La première étape est de récupérer les scripts, disponibles sur GitHub à l'adresse <https://github.com/harfangeek/ansible-scripts>. Il suffit ensuite de se rendre dans le dossier du projet et de lancer Vagrant.

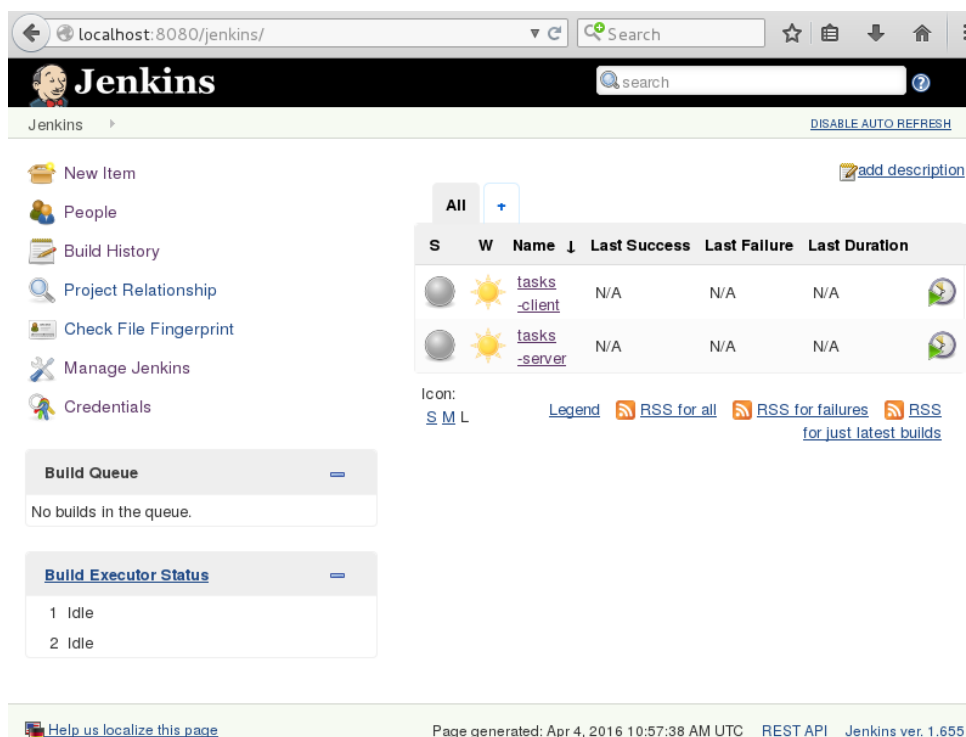
- `git clone https://github.com/harfangeek/ansible-scripts.git`
- `cd ansible-scripts`
- `vagrant up`

La commande `vagrant up` lance la lecture du fichier `Vagrantfile`. Cela permet de configurer une nouvelle VM (image à installer, provisionning etc). Nous détaillerons le contenu de ce fichier dans la partie IV. Dans le cadre de notre projet, la commande va installer une nouvelle machine virtuelle Ubuntu 14.04 64 bits, la configurer (quantité de RAM, redirections de ports, etc), puis exécuter les scripts Ansible. Nous pouvons suivre les étapes du déploiement via Ansible en défilant dans le terminal.

## 3. Utilisation

L'usine logicielle est utilisable dès lors que la machine virtuelle est installée et l'usine déployée. Les différents services exposés par la VM sont disponible à l'adresse <http://localhost:8080/<nom du service>>.

Nous pouvons donc lancer Jenkins en allant sur l'url <http://localhost:8080/jenkins/>.



localhost:8080/jenkins/

Jenkins

DISABLE AUTO REFRESH

New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

Credentials

add description

S	W	Name ↓	Last Success	Last Failure	Last Duration
		<a href="#">tasks -client</a>	N/A	N/A	N/A
		<a href="#">tasks -server</a>	N/A	N/A	N/A

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

Build Queue

No builds in the queue.

Build Executor Status

1 Idle

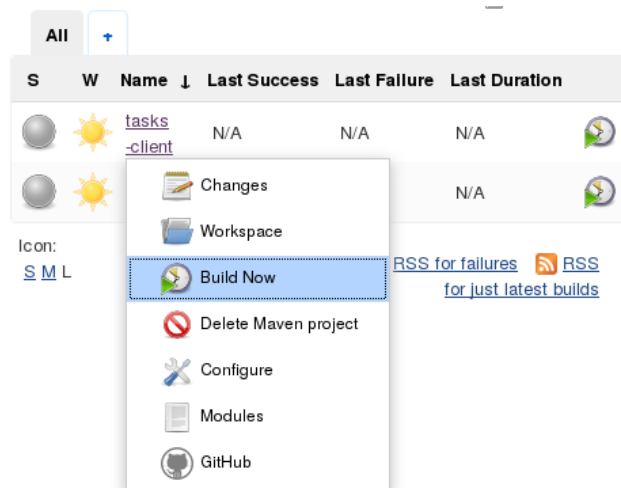
2 Idle

[Help us localize this page](#) Page generated: Apr 4, 2016 10:57:38 AM UTC [REST API](#) Jenkins ver. 1.655

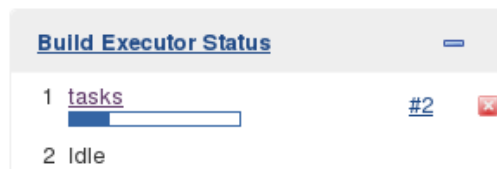
Les jobs *tasks-server* et *tasks-client* sont déjà configurés. Le premier permet de déployer le serveur de l'application et le second déploie le client. Chacun des jobs va récupérer les sources sur GitHub

- <https://github.com/harfangeek/api-tasks-java-ee> pour le serveur
- <https://github.com/harfangeek/tasks-java-ee> pour le client


Puis, les jobs compileront le projet via Maven et déploieront le résultat sur Tomcat. Le job *tasks-server* permet également de déployer la BDD nécessaire pour l'application. A noter que pour déployer un composant, il suffit de cliquer sur la flèche à droite d'un job, puis de cliquer sur « Build » dans le menu.





La progression du build peut être surveillée depuis l'encadré « Build Executor Status ».



Aussi, en cliquant sur le build en cours, nous pouvons voir le log du déploiement :

 **Console Output**

Progress:  

```
Started by user anonymous
Building in workspace /var/lib/jenkins/jobs/tasks
/workspace
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/harfangeek/tasks-java-ee.git # timeout=10
Fetching upstream changes from https://github.com/harfangeek/tasks-java-ee.git
> git --version # timeout=10
> git -c core.askpass=true fetch --tags --progress
https://github.com/harfangeek/tasks-java-ee.git
+refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} #
timeout=10
> git rev-parse refs/remotes/origin/origin
/master^{commit} # timeout=10
Checking out Revision
```

Une fois le déploiement des deux jobs terminés, l'application est disponible via le serveur Tomcat à l'URL <http://localhost:8080/tomcat/tasks-client/>.

Le serveur se trouve à l'URL <http://localhost:8080/tomcat/tasks-server/>.

## 4. Gestion de la machine virtuelle

Vagrant fournit plusieurs commandes pour gérer la machine virtuelle. Il faut se trouver dans le dossier contenant le fichier Vagrantfile de la VM voulu pour pouvoir les utiliser :

- *vagrant provision* : exécuter les scripts ansible une nouvelle fois. Par défaut, les scripts Ansible sont exécutés automatiquement lors de la première mise en route de la VM
- *vagrant ssh* : ouvrir une session ssh sur la VM
- *vagrant halt* : arrêter la VM
- *vagrant reload* : redémarrer la VM

Nous pouvons aussi gérer la VM via VirtualBox notamment pour prendre des snapshots.

## IV. Détail du fichier Vagrantfile

Le fichier Vagrantfile contient la configuration de la VM lancée. Vous trouverez ci-dessous le fichier Vagrantfile commenté :

```
1. vagrant.configure(2) do |config|
2.   # Le nom de la VM
3.   config.vm.box = "usine_logicielle"
4.
5.   # L'adresse à laquelle se trouve l'image de l'OS à utiliser
6.   config.vm.box_url = "https://cloud-images.ubuntu.com/vagrant/trusty/
7.     current/trusty-server-cloudimg-amd64-vagrant-disk1.box"
8.
9.   # Port forwarding, nous redirigeons le port 8080 du host vers le port 80 du guest
10.  config.vm.network "forwarded_port", guest: 80, host: 8080
11.
12.  # Configuration pour utiliser virtualbox comme provider
13.  config.vm.provider "virtualbox" do |vb|
14.    vb.memory = "2048"
15.  end
16.
17.  # Provision avec nos scripts ansible
18.  config.vm.provision "ansible" do |ansible|
19.    ansible.playbook = "install_all.yml"
20.  end
21. end
```

Voici les étapes exécutées :

- Création d'une VM nommée « usine\_logicielle »
- Installation d'une image Ubuntu 14.04 64 bits
- Redirection du port 8080 de la machine host vers le port 80 (apache) de la machine virtuelle.
- Utilisation de Virtualbox comme provider et paramétrage de la VM avec 2048 Mo de RAM.

Enfin, le provisioning est paramétré avec les scripts Ansible.



## V. Détail des scripts Ansible

Dans cette partie, nous revenons en détail sur les étapes du déploiement de l'usine logicielle.

### 1. Script principal : `install_all.yml`

Le script appelé par Vagrant lors du provisionning est `install_all.yml`. Ce dernier est le script principal qui va appeler successivement tous nos scripts.

```
1  ---
2  - hosts: all
3    vars:
4      # http_proxy: "http://147.215.1.189:3128"
5      http_proxy: ""
6      sudo: true
7    tasks:
8      - include: apt.yml http_proxy={{ http_proxy }}
9      - include: git.yml
10     - include: java.yml
11     - include: apache.yml
12     - include: mysql.yml MySQL_root_pass="root"
13     - include: tomcat.yml
14     - include: maven.yml
15     - include: jenkins.yml
```

On retrouve la directive `hosts : all` signifiant que le script s'applique à toutes les machines, la partie `vars` permettant de définir des variables globales au script et la partie `tasks` où toutes les tâches à effectuer sont listée. Ici, nous n'effectuons pas de tâche, nous incluons les autres scripts qui eux contiennent les tâches à réaliser. Nous pouvons passer des variables aux scripts inclus (ici, le proxy à utiliser pour `apt.yml`, et le mot de passe `root` à utiliser pour le scripts `mysql.yml`).

## 2. Mise à jour du cache aptitude avec proxy

```
1 ---
2 - name: update apt cache
3   apt: update_cache=yes
4   environment:
5     http_proxy: "{{ http_proxy }}"
```

Le script *apt.yml* met à jour le cache d'aptitude (équivalent à la commande *apt-get update*). On utilise ici le module *apt* d'Ansible. Aussi, on utilise la directive *environment* qui permet de paramétrer des variables d'environnement pour la tâche en cours. Ici, nous paramétrons le proxy (la valeur est reprise du script parent). On remarque aussi la directive *name* qui permet de mettre un commentaire. Ce commentaire sera affiché lors de l'exécution de la tâche pendant le provisionning.

```
TASK: [update apt cache] **
ok: [default]

TASK: [Install git] *****
ok: [default]

TASK: [install Java] *****
ok: [default]

TASK: [Set Java path] *****
ok: [default]
```

## 3. Git

Le script *git.yml* installe le paquet git via le module *apt*.

```
1 ---
2 - name: Install git
3   apt: name=git state=present
```

## 4. Java

Le script *java.yml* installe le paquet *openjdk-7-jdk* via le module *apt*. Il ajoute ensuite la variable d'environnement *JAVA\_HOME* nécessaire pour les composants utilisant Java.

```
1  ---
2  - name: install Java
3    apt: name=openjdk-7-jdk state=present
4  - name: Set Java path
5    copy:
6      content: "export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/"
7      dest: "~/.bashrc"
```

## 5. Apache

Le script *apache.yml* installe et configure le serveur Apache. Nous pouvons le décomposer en trois étapes.

- Installation :

Le script installe les paquets *apache2* et *libapache2-mod-proxy-html* via le module *apt*. Nous pouvons remarquer la syntaxe permettant de répéter une même opération sur plusieurs éléments (avec la variable `{{item}}` et la clause *with\_items*).

```
1  ---
2  - name: install apache2
3    apt: name={{item}} state=latest
4    with_items:
5      - apache2
6      - libapache2-mod-proxy-html
```

- Activation des modules :

Nous activons ensuite une série de module Apache avec le module `apache2_module`. Cela correspond à la commande `a2enmod`.

```
7 - name: Manage Apache2 modules
8   apache2_module: state=present name={{item}}
9   with_items:
10      - proxy
11      - proxy_http
12      - proxy_ajp
13      - rewrite
14      - deflate
15      - headers
16      - proxy_balancer
17      - proxy_connect
18      - proxy_html
```

- Vhosts :

```
19 - name: Create tomcat and jenkins vhosts
20   template: src=reverse_proxy.conf dest="/etc/apache2/
21             sites-available/reverse_proxy.conf"
22 - name: Enable tomcat and jenkins vhosts
23   command: a2ensite reverse_proxy
24 - name: Restart Apache2
25   service: name=apache2 state=restarted
```

Nous créons un virtual host (`reverse_proxy.conf`) pour paramétrer le reverse proxy pour Tomcat et Jenkins. Nous activons ce virtual host en appelant la commande `a2ensite`. La directive `command` permet d'exécuter une commande sur la machine cible. Enfin nous redémarrons le serveur via le module `service`. Voici le fichier `reverse_proxy.conf` :

```

1. <VirtualHost *:80>
2.     ProxyRequests Off
3.     ProxyPreservehost on
4.     ServerName localhost
5.     ProxyPass /jenkins http://127.0.0.1:9010/jenkins
6.     ProxyPassReverse /jenkins http://127.0.0.1:9010/jenkins
7.     ProxyPass /tomcat http://127.0.0.1:9020/
8.     ProxyPassReverse /tomcat http://127.0.0.1:9020/
9.     AllowEncodedSlashes NoDecode
10.    <Proxy>
11.        Order Allow,Deny
12.        Allow from all
13.    </Proxy>
14. </VirtualHost>

```

Les commandes *ProxyPass* et *ProxyPassReverse* permettent de mettre en place le reverse proxy pour Jenkins et Tomcat. Nous redirigeons les adresses en /tomcat vers le port 9020 et les adresses en /jenkins vers le port 9010.

## 6. MySQL

Le script *mysql.yml* installe et configure MySQL.

```

1  ---
2  - name: Set MySQL root password before installing
3    debconf: name='mysql-server' question='mysql-server/root_password'
4              value='{{MySQL_root_pass | quote}}' vtype='password'
5  - name: Confirm MySQL root password before installing
6    debconf: name='mysql-server' question='mysql-server/root_password_again'
7              value='{{MySQL_root_pass | quote}}' vtype='password'

```

Nous commençons par utiliser le module *debconf* afin de paramétrer le mot de passe root de MySQL. L'installateur de MySQL demandant le mot de passe et la confirmation du mot de passe nous devons d'abord le configurer.

```

8  - name: Install MySQL
9    apt: package={{ item }} state=installed force=yes update_cache=yes
10         cache_valid_time=3600
11    when: ansible_os_family == 'Debian'
12    with_items:
13      - mysql-server
14      - mysql-client
15      - python-mysqldb

```

On installe ensuite les paquets *mysql-server*, *mysql-client* et *python-mysqldb*.

## 7. Tomcat

Le script *tomcat.yml* installe et configure le serveur Tomcat.

```
1  ---
2  - name: install Tomcat
3    apt: name={{item}} state=installed
4    with_items:
5      - tomcat7
6      - tomcat7-admin
7  - name: Change Tomcat server port
8    lineinfile: dest=/etc/tomcat7/server.xml
9                regexp="<Connector port=\"8080\" protocol=\"HTTP/1.1\" \"
10               line="<Connector port=\"9020\" protocol=\"HTTP/1.1\" \"
11  - name: Add tomcat deployer user
12    lineinfile: dest=/etc/tomcat7/tomcat-users.xml
13               regexp="<tomcat-users>"
14               line="<tomcat-users>\n\t<user username=\"deployer\"
15                  |password=\"deployer\" roles=\"manager-script\" />"
16  - name: Restart tomcat
17    service: name=tomcat7 state=restarted
```

Nous commençons par installer les paquets *tomcat7* et *tomcat7-admin*. Puis, nous configurons le port du serveur (9020 au lieu de 8080 par défaut) en modifiant le fichier */etc/tomcat7/server.xml*. Nous utilisons pour cela le module *lineinfile* qui permet de rechercher une chaîne dans un fichier et de la remplacer par une autre. Nous utilisons le même module pour ajouter un nouvel utilisateur, « *deployer* », avec le rôle « *manager-script* » dans le fichier */etc/tomcat7/tomcat-users.xml*. Cet utilisateur nous permettra plus tard de déployer des applications sur Tomcat depuis Jenkins.

## 8. Maven

Le script *maven.yml* installe maven, qui sera nécessaire pour compiler notre application.

```
1  ---
2  - name: install maven
3    apt: name={{item}} state=latest force=yes
4    with_items:
5      - maven
```

## 9. Jenkins

Le script *jenkins.yml* installe et configure Jenkins. Premièrement nous installons Jenkins via le module *apt*, en ajoutant tout d'abord le dépôt Jenkins.

```
1  ---
2  - name: Add jenkins repository key
3    apt_key:
4      url='http://pkg.jenkins-ci.org/debian/jenkins-ci.org.key'
5      state=present
6  - name: Add jenkins repository
7    apt_repository:
8      repo='deb http://pkg.jenkins-ci.org/debian binary/'
9      update_cache=yes
10 - name: Install jenkins
11   apt:
12     name=jenkins
13     state=present
```

Ensuite, nous configurons le nom de l'instance Jenkins, le numéro de port (9010 au lieu du port par défaut) ainsi que les arguments utilisé au lancement de l'instance. Pour ces trois étapes nous utilisons le module *lineinfile* pour modifier le fichier */etc/default/jenkins*.

```
14 - name: Change jenkins server name
15   lineinfile: dest=/etc/default/jenkins
16               regexp="NAME="
17               line="NAME=jenkins"
18 - name: Change jenkins server port
19   lineinfile: dest=/etc/default/jenkins
20               regexp="HTTP_PORT="
21               line="HTTP_PORT=9010"
22 - name: Change jenkins server arguments
23   lineinfile: dest=/etc/default/jenkins
24               regexp="JENKINS_ARGS="
25               line="JENKINS_ARGS=\"--webroot=/var/cache/$NAME/war
26               --httpPort=$HTTP_PORT --ajp13Port=$AJP_PORT --prefix=$PREFIX\""
```

On redémarre ensuite le service Jenkins afin de prendre en compte les précédentes configurations. La deuxième tâche va attendre que le service soit totalement démarré. Pour cela, nous faisons des requêtes sur le serveur Jenkins jusqu'à ce qu'il retourne un code HTTP 200.

```
27 - name: Restart jenkins
28   service: name=jenkins state=restarted
29 - name: Wait until Jenkins is available
30   shell: curl --head --silent http://localhost:9010/jenkins/
31   register: result
32   until: result.stdout.find("200 OK") != -1
33   retries: 12
34   delay: 5
```

Nous devons maintenant configurer les jobs Jenkins pour le serveur et le client de notre application J2EE. Les jobs sont configurés dans le dossier `/var/lib/jenkins/jobs`. Chaque job se compose d'un dossier portant le nom du job et contenant un fichier `config.xml` où se trouve sa configuration. Nous avons préparé deux jobs que nous avons archivé (`tasks-server.tar.gz` et `tasks-client.tar.gz`). Pour le déploiement, nous copions donc ces archives sur la machine virtuelle, puis nous les décompressons dans le dossier `/var/lib/jenkins/jobs`. La dernière étape consiste à paramétrer le propriétaire de ces dossiers.

```
35 - name: Copy jenkins job
36   copy: src={{item}}.tar.gz dest=/root/
37   with_items:
38     - tasks-client
39     - tasks-server
40 - name: Extract jenkins job
41   command: tar xzvf /root/{{item}}.tar.gz -C /var/lib/jenkins/jobs/
42   with_items:
43     - tasks-client
44     - tasks-server
45 - name: Set jenkins job owner
46   command: chown -R jenkins:jenkins /var/lib/jenkins/jobs/{{item}}
47   with_items:
48     - tasks-client
49     - tasks-server
```

Pour la suite, nous installons les plugins nécessaires à nos déploiements. Nous récupérons tout d'abord l'archive `java jenkins-cli.jar` qui permettra ensuite l'installation des plugins en ligne de commande.

```
50 - name: Get jenkins-cli.jar
51   get_url:
52     url=http://localhost:9010/jenkins/jnlpJars/jenkins-cli.jar
53     dest=/root/
54 - name: Install Jenkins plugins
55   command: java -jar /root/jenkins-cli.jar -s http://localhost:9010/
56             jenkins install-plugin {{item}}
57   with_items:
58     - git
59     - github
60     - checkstyle
61     - crap4j
62     - dry
63     - htmlpublisher
64     - jdepend
65     - pmd
66     - violations
67     - xunit
68     - clover
69     - deploy
```

Pour finir, nous configurons le plugin Maven, en lui indiquant où se trouve notre installation de Maven. Nous redémarrons enfin le serveur.



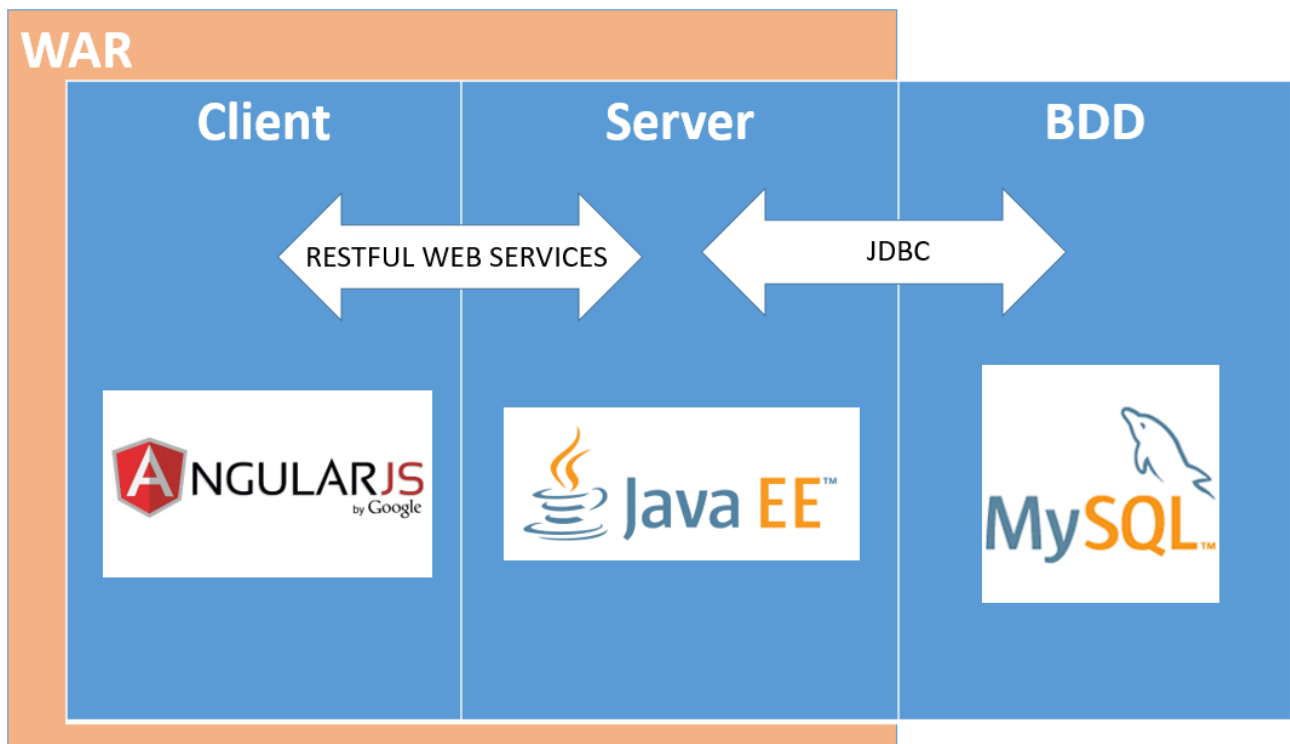
```
70 - name: Configure Maven in Jenkins
71   template: src=jenkins_maven.xml dest=/var/lib/jenkins/hudson.tasks.Maven.xml
72 - name: Restart jenkins
73   service: name=jenkins state=restarted
```

Le fichier de configuration du plugin Maven se trouve au chemin  
/var/lib/jenkins/hudson.tasks.Maven.xml. Le voici :

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <hudson.tasks.Maven_-DescriptorImpl>
3    <installations>
4      <hudson.tasks.Maven_-MavenInstallation>
5        <name>default</name>
6        <home>/usr/share/maven</home>
7        <properties/>
8      </hudson.tasks.Maven_-MavenInstallation>
9    </installations>
10 </hudson.tasks.Maven_-DescriptorImpl>
```

Nous précisons au plugin que notre installation de Maven se trouve dans le dossier  
/usr/share/maven.

## VI. Détail de l'appli



# Lexique

---

**Usine logicielle** : Ensemble d'outils permettant la production de logicielles. Ces outils forment une chaîne de production automatisant de nombreuses tâches du cycle de vie d'un logiciel. Une usine logicielle se compose d'outil de gestion de code collaboratif, de gestion de dépendance, de chaînes de compilations, d'analyse de la qualité, et du déploiement vers divers environnements.

**Provisionnement** : Déploiement sur un environnement (physique ou virtuel).

**Script** : Fichier contenant un ensemble d'instructions à exécuter.

**Paquet** : Archive contenant un ensemble d'exécutables et de fichiers de configuration permettant d'installer un logiciel.

**Repository** : Dans le cadre d'un gestionnaire de code source, serveur de dépôt sur laquelle plusieurs développeurs collaborent en y déposant leurs fichiers sources. Dans le cadre de l'installation de paquet, serveur sur lequel est regroupé un ensemble de paquets téléchargeable par les utilisateurs.

**Audit de code** : Gestion de la qualité du code sources. La qualité peut être évaluée en définissant des règles (norme de nommage) et en vérifiant si ces règles sont respectées. L'audit peut aussi se faire en analysant l'exécution du programme et en cherchant d'éventuels comportements inattendus (fuite de mémoire).

**Build** : Dans le cadre de Jenkins, ensemble d'étapes permettant le déploiement d'une application. Ces étapes sont par exemple la récupération du code source, la compilation et le déploiement sur un serveur.

## Webographie

---

<http://blog.erlem.fr/>

<https://git-scm.com/documentation>

<https://httpd.apache.org/docs/>

[tomcat.apache.org/tomcat-7.0-doc/](http://tomcat.apache.org/tomcat-7.0-doc/)

[maven.apache.org/guides/](http://maven.apache.org/guides/)

<https://wiki.jenkins-ci.org/>

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

## Conclusion

---

Ce projet a consisté en la création d'une usine logicielle J2EE. Celle-ci permet à une équipe de développeurs de travailler en déploiement continue.

Pour certains membres de l'équipe, ce projet fût d'une part l'occasion de découvrir des technologies telles que Jenkins, Maven ou Nexus et d'autre part, d'apprendre de nouveaux concepts tels que le DevOps.

Pour d'autres, ce projet a permis d'approfondir et de solidifier leurs connaissances sur les outils vues dans le cadre du projet.

En somme, l'unité SI5I21 a été très enrichissante de par sa complexité et la variété des logiciels utilisés.