

## **Jenkins, mettre en place l'intégration continue en java**

<b>Objectifs pédagogiques</b>	<b>3</b>
Où trouver les ressources du cours	3
<b>Pré-requis</b>	<b>4</b>
<b>Installation des outils</b>	<b>5</b>
Sous Linux/Ubuntu	5
Sous Windows	6
1- Installation via scoop( <a href="https://scoop.sh/">https://scoop.sh/</a> ) ou choco( <a href="https://chocolatey.org/">https://chocolatey.org/</a> )	6
<b>Notions de gestion de projets</b>	<b>8</b>
• Les besoins, idées, problématique	8
• Une Organisation	8
• L'architecture de l'application ou du module	8
• Les ressources	9
• La réalisation de l'application	9
• Le marketing	9
<b>Les 12 Facteurs/ The Twelve-Factor App</b>	<b>10</b>
<b>Le Générateur de projet web java jhipster</b>	<b>13</b>
Génération d'un projet jhipster et déploie sur heroku(provider cloud avec une offre free)	13
Déployer le projet sur github	15
Quelques liens utiles	16
<b>L'intégration continue et le déploiement continue (CICD)</b>	<b>17</b>
<b>Jenkins</b>	<b>21</b>
Introduction	21
Un bref rappel historique	21
Installation de jenkins	23
Lancement de jenkins via le navigateur	25
> 127.0.0.1:8080	25
Présentation de l'interface Jenkins	30

Création de notre premier job en mode free style	31
Exécution de mon premier job	33
<b>Annexe 1 : Introduction à docker</b>	<b>39</b>
Définition	39
Qu'est-ce qu'une image Docker?	42
Qu'est-ce qu'un conteneur Docker?	43
Conclusion	43
Les commandes de base à connaître	43
Exemples:	45

# Objectifs pédagogiques

À l'issue de la formation, le participant sera en mesure de :

- Comprendre les principes de l'intégration continue en vue de son implémentation;
- Intégrer Jenkins avec les autres outils (SCM, gestionnaire de tickets...);
- Mettre en place un serveur Jenkins automatisant les build;
- Automatiser les tests, les audits de code et les déploiements sur la plateforme d'intégration Jenkins

## Où trouver les ressources du cours

Les ressources de cours se trouve sur le dépôt github suivant

<https://github.com/e-metconsulting/client-orsys-cours> :

Pour récupérer les sources en local, lancez la commande suivante si vous avez git sur votre pc.

```
git clone git@github.com:e-metconsulting/client-orsys-cours.git
```

# Pré-requis

Pour aborder sereinement les modules de ce cours, la connaissance du langage Java et des notions du cycle de développement (Maven) seront capitales sans toutefois être un frein majeur.

La connaissance de certains outils listés ci-dessous serait un plus.

- Un environnement linux : Ubuntu, Fedora
- Si possible connaître **git** et avoir un compte sur <https://github.com/> et ou <https://gitlab.com/> et être familier aux notions de branch et de repository etc
- Si possible avoir des notions de docker

# Installation des outils

## Sous Linux/Ubuntu

- **Visual Studio Code**, Sublime-text
- **Git**: `sudo apt install git`
- **Git-flow**: `sudo apt-get install git-flow`
- **Création d'un compte github**:
- **Génération et Ajout de la clé ssh**: `ssh-keygen -o -t rsa`
- **Maven**: `sudo apt install maven`
- **Open jdk 8**: `sudo apt install openjdk-8-jdk`
- **Open jdk 11**: `sudo apt install openjdk-11-jdk`
  - `update-alternatives --list java`
- **NodeJs & NPM**: `sudo apt install nodejs npm`
- **yarn**: `sudo apt install yarn`
- **Jhipster Generator**: `sudo npm install -g generator-jhipster`
- **angular/cli**: `npm install -g @angular/cli`
- **Docker**:
  - `sudo apt-get update`
  - `sudo apt-get remove docker docker-engine docker.io`
  - `sudo apt install docker.io`
  - `sudo systemctl start docker`
  - `sudo systemctl enable docker`
  - `docker --version`
  - `sudo apt install curl`
  - `sudo curl -L https://github.com/docker/compose/releases/download/1.24.1/docker-compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose`
  - `sudo chmod +x /usr/local/bin/docker-compose`
  - `docker-compose --version`

## Sous Windows

1- Installation via scoop(<https://scoop.sh/>) ou choco (<https://chocolatey.org/>)

- <https://github.com/lukesampson/scoop/wiki>
- <https://github.com/lukesampson/scoop/wiki/Quick-Start>
- **Configuration local**
  - `$env:SCOOP='C:\DEV\Tools\scoop'`
  - `[environment]::setEnvironmentVariable('SCOOP',$env:SCOOP,'User')`
  - exemple : `scoop install curl`
- **Configuration global:**
  - `$env:SCOOP_GLOBAL='c:\apps'`
  - `[environment]::setEnvironmentVariable('SCOOP_GLOBAL',$env:SCOOP_GLOBAL,'Machine')`
  - Exemple: `scoop install -g pandoc`
- **Recherche d'un install**
  - `Scoop search maven`
  - `scoop info maven`
- **Installation:**
  - `scoop bucket add extras`
  - `scoop bucket add java`
  - `scoop install` `vscode`  
<https://code.visualstudio.com/docs/languages/java>
  - `scoop install idea (intellij)`
  - `scoop install eclipse-jee`
  - `scoop info sublime-text`
  - `scoop install cmdr`
  - `scoop install git`
  - Installation des jdk
    - `scoop bucket add java`
    - `scoop install ojdkbuild8/adopt8-upstream/openjdk11`
    - <https://adoptopenjdk.net/>

- <https://www.oracle.com/java/technologies/javase-downloads.html>
- scoop install maven
- scoop install nodejs / scoop install nodejs-lts
  - scoop install nvs
  - scoop install nvm
- scoop install yarn
- npm install -g generator-jhipster
- npm install -g @angular/cli
  - <https://cli.angular.io/>
- Docker
  - Boot2docker
  - Docker Toolbox
    - <https://blog.lecacheur.com/2015/10/14/docker-au-revoir-boot2docker-bonjour-docker-toolbox/>
  - Scoop
    - <https://github.com/lukesampson/scoop/wiki/Docker>
  - Docker Desktop
    - <https://docs.docker.com/docker-for-windows/install-windows-home>
    - <https://docs.microsoft.com/en-us/windows/wsl/install-win10>
    - <https://docs.microsoft.com/fr-fr/windows/wsl/wsl2-kernel>
    - <https://docs.docker.com/docker-for-windows/install/>
  - Running docker :
    - docker-compose -f src\main\docker\postgresql.yml up -d

# Notions de gestion de projets

- ***Les besoins, idées, problématique***
- ***Une Organisation***
  - Cycles en V
    - Cahier de charges
    - Analyse fonctionnelle et Analyse technique
    - les composant technique et leurs communication
  - Une Agilité
    - Scrum
    - User story
  - Hybride (Lean)
- ***L'architecture de l'application ou du module***
  - MVC
  - Client lourd fort impact lors du déploiement
    - Synchronisation complexe pour l'arrêt et donc les mise à jours ( application 24/24)
    - Sensibilité accrue à mettre en place des règles de rollback en cas d'échec.
    - Monolithique
    - La puissance de calcul est sur le serveur du client
    - La puissance de calcul sur la machine du client
  - Client léger faible impact lors du déploiement.
    - Déploiement continue
    - Time to market rapide
    - Réactive
    - Micro service
    - Serverless
    - Application Web



- ***Les ressources***

- Serveurs
- Reseaux
- Protocoles de sécurités
- Les certificats
- Les licences
- La scalabilité (supporter la charge)
- etc

- ***La réalisation de l'application***

- Application monolithe
- Application micro service (conteneur)
- Application Web
- Application mobile
- Gestion de la sécuriser (transverse)
- Mise en place des certificats java
- comment la faire communiquer,
- La distribution, le déploiement, la livraison de l'application (diffuser ses mises à jour)
- Maintenance
- Son obsolescence

- ***Le marketing***

# Les 12 Facteurs/ The Twelve-Factor App

Les 12 Facteurs définissent les 12 règles qui doivent régir le développement d'une application (<https://12factor.net/fr/>)

À l'époque actuelle, les logiciels sont régulièrement délivrés en tant que **services** : on les appelle des applications **web** (web apps), ou logiciels en tant que service (**software-as-a-service**).

L'application 12 facteurs est une méthodologie pour concevoir des logiciels en tant que service qui :

- Utilisent **des formats déclaratifs** pour mettre en oeuvre l'automatisation, pour minimiser le temps et les coûts pour que de nouveaux développeurs rejoignent le projet;
- Ont un **contrat propre** avec le système d'exploitation sous-jacent, offrant une **portabilité maximum** entre les environnements d'exécution;
- Sont adaptés à des **déploiements** sur des plateformes cloud modernes, rendant inutile le besoin de serveurs et de l'administration de systèmes;
- **Minimisent la divergence** entre le développement et la production, ce qui permet le **déploiement continu** pour une agilité maximum;
- et peuvent **grossir verticalement** sans changement significatif dans les outils, l'architecture ou les pratiques de développement;

La méthodologie 12 facteurs peut être appliquée à des applications écrites dans tout langage de programmation, et qui utilisent tout type de services externes (base de données, file, cache mémoire, etc.)

Ils concernent tout développeur qui construit des applications qui fonctionnent en tant que service, ainsi que les personnes qui déploient et gèrent de telles applications.

1. **Base de code**: Le code à un endroit et versionné (git)

2. **Dépendances:** Outil de gestion des dépendances (maven, composer, npm etc)
3. **Configuration:** Avoir un fichier de configuration pour tous les environnements et recourir aux variables d'environnement pour spécifier les valeurs (user, ip, database) exemple docker-compose
4. **Services Externes:** Les services doivent être gérés comme des éléments externes, exemple les connecteurs de base de données
5. **Build/Release/Run:** Ne pas les mélanger dans les pipelines de CI/CD
6. **Processus:** Sans état (stateless) doit permettre la scalabilité horizontales
7. **Association de ports:** Les services communiquent sur un port
8. **Concurrence:** La scalabilité horizontale c'est à dire création de processus plutôt que l'augmentation de la CPU/RAM
9. **Jetable:** Arrêt gracieux et relance sans impact; doit savoir où elle en est

10. **Parité développement production:** intégration des développeurs aux déploiements
11. **Logs:** flux d'évènements (logstash, elk)
12. **Processus d'administration:** code admin et applicatif ensemble pas de scission

# Le Générateur de projet web java jhipster

Jhipster est un générateur d'application libre open source très prisé et populaire; utilisé pour le développement d'applications web modernes en utilisant Angular/React/Kotlin et le framework Spring.

C'est donc une plateforme full stack permettant de créer aussi bien des applications monolithiques que des applications respectant l'architecture de micro services .

Il est nativement responsive design et déployable sur le cloud comme en on-premise sur une vm dédiée et facilement scalable via micro services exposant une api rest. C'est donc une application maven complète

***Génération d'un projet jhipster et déploiement sur heroku(provider cloud avec une offre free)***

- > création du dossier **jhdemo**
- > lancer la commande jhipster
- > répondez aux questions par défaut en générant un projet monolithique

```
INFO! Using JHipster version installed globally
INFO! Executing jhipster:app

JHIPSTER

https://www.jhipster.tech

Welcome to JHipster v6.10.4
Application files will be generated in folder: /home/bilonjea/Desktop/testo

Documentation for creating an application is at https://www.jhipster.tech/creating-an-app/
If you find JHipster useful, consider sponsoring the project at https://opencollective.com/generator-jhipster

JHipster update available: 6.10.5 (current: 6.10.4)

Run npm install -g generator-jhipster to update.

? Which *type* of application would you like to create? Monolithic application (recommended for simple projects)
? [Beta] Do you want to make it reactive with Spring WebFlux? No
? What is the base name of your application? prosoccer
? What is your default Java package name? com.pro.soccer.com
? Do you want to use the JHipster Registry to configure, monitor and scale your application? No
? Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
? Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
? Which *production* database would you like to use? PostgreSQL
? Which *development* database would you like to use? PostgreSQL
? Do you want to use the Spring cache abstraction? Yes, with the Ehcache implementation (local cache, for a single node)
? Do you want to use Hibernate 2nd level cache? Yes
? Would you like to use Maven or Gradle for building the backend? Maven
? Which other technologies would you like to use?
? Which *Framework* would you like to use for the client? Angular
? Would you like to use a Bootswatch theme (https://bootswatch.com/)? Flatly
? Choose a Bootswatch variant navbar theme (https://bootswatch.com/)? Light
? Would you like to enable internationalization support? Yes
? Please choose the native language of the application French
? Please choose additional languages to install English
? Besides JUnit and Jest, which testing frameworks would you like to use? Protractor
? Would you like to install other generators from the JHipster Marketplace? No

Installing languages: fr, en
```

Fig : Installation de jhipster

## Déployer le projet sur github

- > Création d'un repo sur github et le linker avec son repo local
- >git init
- >git add .
- >git commit -m "first commit"
- >git remote add origin
- [git@github.com](mailto:git@github.com):login/jhdemo.git
- >git push -u origin master
  
- > builder le projet
  - > maven clean install
  - > lancer le docker de la base données : docker-compose -f src/main/docker/postgres.yml up
  - > lancer l'application jhipster via java -jar target/monapplication.jar
- > Intégration du projet sur heroku <https://dashboard.heroku.com/>
  - > Création d'un nouvelle application exemple : jhdemovm
  - > Sélectionnez la méthode de déploiement github
  - > Autorisez votre compte github sur heroku
  - > Sélectionner l'application jhipster à déployer
  - > cliquez sur connecter
  - > Sélectionnez la branch de déploiement
  - > Cliquez sur Deploy Branch
  - > à la fin du déploiement vous aurez l'url de l'application
- > intégration d'un schéma avec jdl <https://www.jhipster.tech/jdl/>
  - > jdl studio (<https://start.jhipster.tech/jdl-studio/>)
  - > télécharger le fichier jdl dans votre projet dans src/main/jdl
  - > générer vos entités avec la commande : jhipster import-jdl jhipster-jdl.jh

Pour gagner du temps il est également possible de générer le projet jhipster en ligne directement dans son repo github ou gitlab en se rendant ici : <https://start.jhipster.tech/>

## Quelques liens utiles

> Présentation de jhipster en 5 min :

<https://www.jhipster.tech/presentation/#/>

> Site officiel : <https://www.jhipster.tech/>

> Un exemple de génération de projet :

<https://www.bearstudio.fr/blog/jhipster-generateur-projet-hipsters>

Ce projet jhipster nous servira de base pour mettre en place les notions de CI/CD (Continuous integration Continuous Deployment) via Jenkins.

Il sera également possible de générer rapidement un petit projet maven spring-boot admin via l'outil spring <https://start.spring.io/>.

Nous sommes enfin outillés pour aborder Jenkins pour ainsi dire .



# L'intégration continue et le déploiement continue (CICD)

L'intégration continue (CI) est une pratique logicielle qui nécessite la validation fréquente du code dans un référentiel partagé.

La validation du code détecte plus souvent les erreurs plus tôt et réduit la quantité de code dont un développeur a besoin pour déboguer afin de trouver la source d'une erreur.

Les mises à jour fréquentes du code facilitent également la fusion des modifications de différents membres d'une équipe de développement logiciel.

C'est idéal pour les développeurs, qui peuvent passer plus de temps à écrire du code et moins de temps à déboguer les erreurs ou à résoudre les conflits de merge.

Lorsque vous validez du code dans votre référentiel, vous pouvez en permanence créer et tester le code pour vous assurer que la validation n'introduit pas d'erreurs.

Vos tests peuvent inclure des linters de code (qui vérifient la mise en forme du style), des contrôles de sécurité, la couverture du code via les tests unitaires, des tests fonctionnels et de charges et d'autres contrôles personnalisés.

Un outil tel que Sonar, assure la validation de la qualité des livrables avec un suivi régulier par l'implémentation des règles qualitatives.

Le “**building**” et le **test** de votre code nécessitent un **serveur**. Vous pouvez créer et tester les mises à jour localement avant de pousser le code vers un référentiel (repository manager tel que github, gitlab ou bitbucket etc .),

ou vous pouvez utiliser un serveur **CICD** (tel jenkins, github ou gitlab, bamboo par exemple ) qui vérifie les nouveaux commits de code dans un **repository**.

En ce qui concerne le déploiement continue, Il s'agira de pousser ou déployer régulièrement les nouvelles versions de notre application de façon automatisée dans les environnements cibles.

Cette notion de déploiement continue peut s'étendre également au fait de stocker dans un référentiel tel que nexus les artefacts de nos composants applicatifs ou nos images dockers dans une registry tel que docker hub ou gitlab registry etc.

Pour mettre en oeuvre votre intégration et déploiement continue, il faut plusieurs serveurs:

- Un serveur de sources du projet exemple github (git)
- Un serveur pour gérer ses pipelines d'intégration et de déploiement (jenkins, gitlab, github)
- Un repository pour sauvegarder les artefacts de ses projets (nexus, artifactory, github, maven repository ...)
- Un serveur pour la qualité de codes, sonar par exemple
- Des serveurs pour déployer son application pour le test, pour la production, des vps chez des providers cloud ou non, des machines locales, des conteneurs dockers etc.

Il existe donc plusieurs sources de données (serveurs) à provisionner, gravitant autour de son application. Cela demande donc la mise en place d'une architecture dédiée à l'intégration continue et au déploiement continu.

C'est donc au système d'intégration continue et de déploiement que revient la tâche d'orchestrer les workflows et ou pipeline pour provisionner et consommer les données des différents tiers.

Ainsi, il existe différents systèmes mettant en œuvre le concept d'orchestration du workflow pour l'intégration continue et le déploiement continu. On peut citer par exemples

- Azure pipelines
- CircleCI
- Gitlab CI/CD
- Travis CI
- Bamboo
- GitHub Actions
- **Jenkins**

Il est généralement simple avec la documentation fournie de passer d'un orchestrateur à un autre aisément. Cet orchestrateur peut s'appuyer sur un ensemble d'outils tels que Ansible, Consul, des workers pour entendre ses fonctionnalités.

Par la suite, nous porterons notre attention sur jenkins. Il s'agira de présenter dans les grandes lignes cet orchestrateur de CI/CD avec un bref aperçu historiques.

Il sera question également de présenter les différentes façons d'installer jenkins tout en privilégiant l'usage de docker. Une fois l'installation de jenkins réalisées nous présenterons son interface, quelques éléments de configurations et nous créerons "from scratch", nos premiers jobs, installerons quelques plugins pour la gestions rapides des utilisateurs et des droits dans un premier temps.

Par la suite nous aborderons des notions de plus en plus avancées que nous citons sans être exhaustifs tels que:

- L'utilisation de git, github et maven
- Le déclenchement automatique d'un job (cascading, triggers, scrutation etc.
- Le jenkinsFile

- Les pipelines
- Les pipelines multibranch
- La configuration et administration avancées de jenkins
- Les scripts groovy
- Le test, déploiement
- etc.

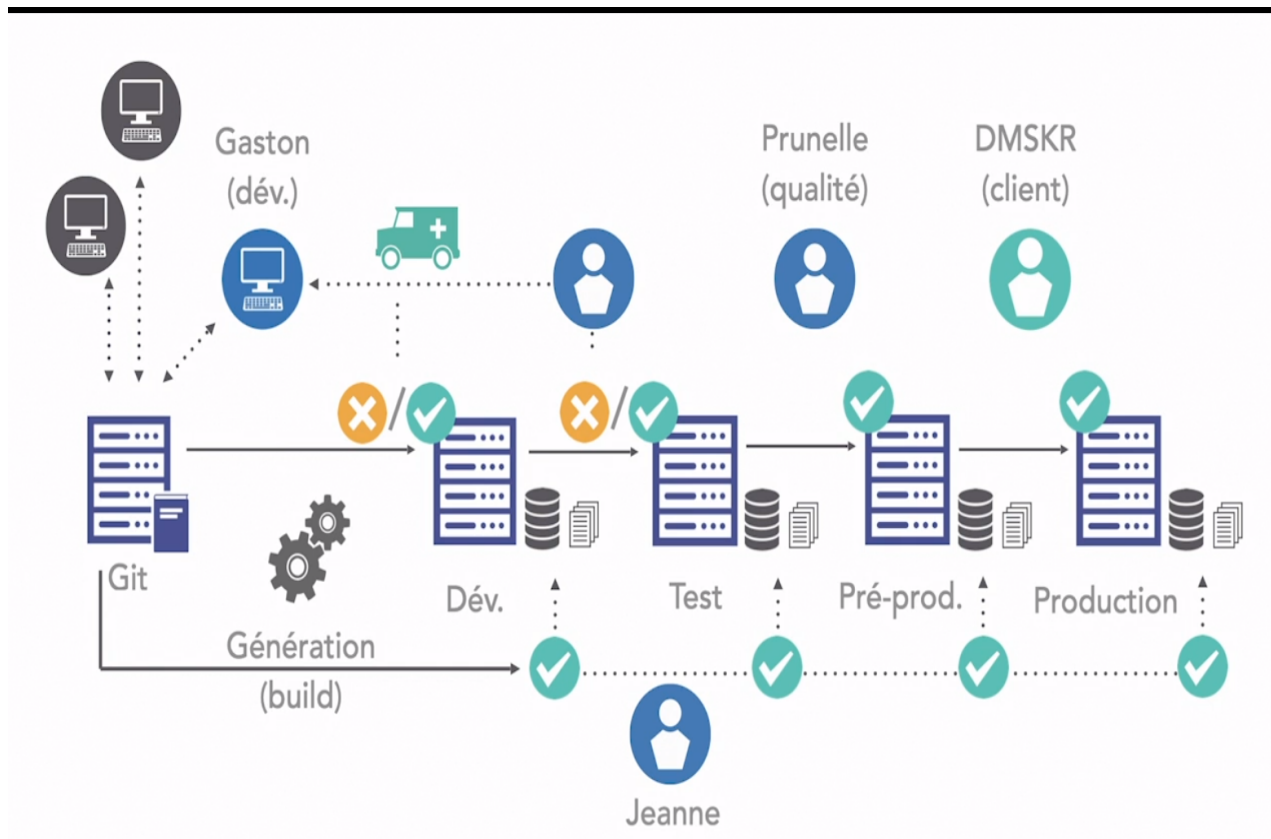


Fig. passer du build à la livraison continue

# Jenkins

## Introduction

Jenkins est un outil open source de serveur d'automatisation. Il aide à automatiser les parties du développement logiciel liées au build, aux tests et au déploiement, et facilite l'intégration continue et la livraison continue.

Écrit en Java, Jenkins fonctionne dans un conteneur de servlets tel qu'Apache Tomcat, ou en mode autonome avec son propre serveur Web embarqué.

Il s'interface avec des systèmes de gestion de versions tels que CVS, Git et Subversion, et exécute des projets basés sur Apache Ant et Apache Maven aussi bien que des scripts arbitraires en shell Unix ou batch Windows.

Ses fonctionnalités peuvent être étendues facilement à l'aide de plugins supplémentaires qui vous permettront, par exemple, de connecter l'outil à d'autres environnements de test.

Les générations de projets peuvent être amorcées par différents moyens, tels que des mécanismes de planification similaires au cron, des systèmes de dépendances entre générations, ou par des requêtes sur certaines URL spécifiques. (wikipedia)

## Un bref rappel historique

Jenkins était à l'origine nommé Hudson et fut renommé en 2011 après des différends entre son auteur, Kohsuke Kawaguchi, et Oracle qui avait fait un fork du projet et revendiquait les droits sur le nom du projet.

La branche d'Oracle, Hudson, continua d'être développée pendant un temps avant d'être donnée à la fondation Eclipse.

Hudson n'est plus maintenu à jour et est annoncé obsolète en février 2017. Autour de 2008, Hudson est devenu une solution de remplacement populaire à l'outil de référence CruiseControl.

Le 11 janvier 2011, une proposition pour renommer Hudson a été annoncée afin d'éviter des problèmes avec un éventuel enregistrement (marque déposée) du nom par Oracle.

Après l'échec des négociations avec Oracle un vote en faveur du renommage a été entériné le 29 janvier 2011.

Le 20 avril 2016, la version 2.0 est mise en ligne avec le plugin Pipeline activé par défaut. Ce plugin permet la rédaction d'instructions de Build utilisant un langage de domaine spécifique basé sur Apache Groovy. (wikipedia)

Pour résumer Jenkins est un ordonnanceur Scheduler permettant d'automatiser les tâches, les programmer, les tester, les vérifier, les enchaîner. Il permet de lancer des jobs ou tâches de façon intelligente, de créer des pipelines, c'est-à-dire des tunnels d'intégration continue et de déploiement continue.

Jenkins dispose de plus de 1000 plugins, très simple à installer (mais la maintenance sera la maintenance de ceci). Ainsi on trouve des plugins pour tous les usages, par exemple pour améliorer la gestion, l'interface, faire tourner les jobs plus facilement ou sur des outils particuliers tel que Docker.

Il possède une interface graphique, mais peut également s'utiliser en ligne de commande via un CLI.

# Installation de jenkins

L'installation de jenkins peut se faire de diverses manières selon les systèmes d'exploitation.

Les environnements linux sont globalement utilisés pour instancier et manager des applicatifs tel que jenkins. C'est ainsi que nous travaillerons essentiellement sur une distribution linux tel qu' ubuntu.

La documentation d'installation de jenkins se trouve dans le site jenkins.io à la page : <https://www.jenkins.io/doc/book/installing/>

[> User Documentation Home](#)

**User Handbook**

- [User Handbook Overview](#)
- **[Installing Jenkins](#)**
  - [Docker](#)
  - [Kubernetes](#)
  - [Linux](#)
  - [macOS](#)
  - [WAR files](#)
  - [Windows](#)
  - [Other Systems](#)
  - [Offline Installations](#)
  - [Initial Settings](#)
- [Using Jenkins](#)
- [Pipeline](#)
- [Blue Ocean](#)
- [Managing Jenkins](#)
- [Securing Jenkins](#)
- [System Administration](#)
- [Scaling Jenkins](#)
- [Troubleshooting Jenkins](#)
- [Glossary](#)

**Tutorials**

- [Guided Tour](#)
- [Jenkins Pipeline](#)
- [Using Build Tools](#)

**Resources**

- [Pipeline Syntax reference](#)
- [Pipeline Steps reference](#)
- [LTS Upgrade guides](#)

[< User Handbook Overview](#)

[Index](#)

[Docker >](#)

## Installing Jenkins

The procedures in this chapter are for new installations of Jenkins.

Jenkins is typically run as a standalone application in its own process with the built-in [Java servlet](#) container/application server ([Jetty](#)).

Jenkins can also be run as a servlet in different Java servlet containers such as [Apache Tomcat](#) or [GlassFish](#). However, instructions for setting up these types of installations are beyond the scope of this chapter.

**Chapter Sub-Sections**

- [Docker](#)
- [Kubernetes](#)
- [Linux](#)
- [macOS](#)
- [WAR files](#)
- [Windows](#)
- [Other Systems](#)
- [Offline Installations](#)
- [Initial Settings](#)

[< User Handbook Overview](#)

[Index](#)

[Docker >](#)

[Was this page helpful?](#)

Fig: documentation d'installation de jenkins

- installation par paquet

<https://pkg.jenkins.io/debian-stable/>

●

```
wget -q -O -http://pkg.jenckins-ci.org/debian/jenkins-ci.org.key | sudo  
apt-key add -
```

```
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ >  
/etc/apt/sources.list.d/jenkins.list'
```

●

```
sudo apt-get update  
sudo apt-get install jenkins
```

Attention à la clef nécessaire à la première connection de l'admin ce  
password se trouve dans les logs ici :

`/var/jenkins_home/secrets/initialAdminPassword`

- Installation par docker-compose

**version: '2'**

**services:**

**jenkins:**

**image: 'jenkins/jenkins:lts'**

**labels:**

**kompose.service.type: nodeport**

**ports:**

**- '80:8080'**

**- '443:8443'**

**- '50000:50000'**

**volumes:**

**- 'jenkins\_data:/jenkins\_config'**

**volumes:**



**jenkins\_data:**  
**driver: local**

Lancement du docker: **docker-compose up -d**

Vérification du run : **docker ps**

Ces commandes doivent être lancées avec un sudo si on a pas défini de user docker.

```
sudo usermod -aG docker $USER  
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose  
sudo service docker restart  
systemctl status docker.service
```

## **Lancement de jenkins via le navigateur**

> 127.0.0.1:8080

## Débloquer Jenkins

Pour être sûr que Jenkins soit configuré de façon sécurisée par un administrateur, un mot de passe a été généré dans le fichier de logs ([où le trouver](#)) ainsi que dans ce fichier sur le serveur :

`/var/jenkins_home/secrets/initialAdminPassword`

Veillez copier le mot de passe depuis un des 2 endroits et le coller ci-dessous.

Mot de passe administrateur



Continuer

Fig : débloquent Jenkins avec la clé de connexion

Cette commande permet d'avoir de récupérer la clé dans les logs **`docker logs beb254914277`** où **`beb254914277`** est l'id du docker Jenkins via **`docker ps`**.

On peut également lancer la commande à partir du nom de l'instance Docker : **`docker logs jenkins_jenkins_1`**

Il est également possible de récupérer la clé depuis le lien indiqué ci-dessus :  
En se connectant au Docker :

- `sudo docker exec -it beb254914277 sh`
- `cat /var/jenkins_home/secrets/initialAdminPassword`

# Personnaliser Jenkins

Les plugins étendent Jenkins avec des fonctionnalités additionnelles pour satisfaire différents besoins.

## Installer les plugins suggérés

Installer les plugins que la communauté Jenkins trouve les plus utiles.

## Sélectionner les plugins à installer

Sélectionner et installer les plugins les plus utiles à vos besoins.

Fig: Installation des de bases

Installation en cours...

# Installation en cours...

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding	<b>Folders</b> ** JavaBeans Activation Framework (JAF) API ** JavaMail API ** bouncycastle API ** Instance Identity ** Mina SSHD API :: Common ** Mina SSHD API :: Core ** SSH server <b>OWASP Markup Formatter</b> ** Structs ** Token Macro <b>Build Timeout</b> ** Credentials ** Trilead API ** SSH Credentials ** Pipeline: Step API ** Plain Credentials <b>Credentials Binding</b> ** SCM API ** Pipeline: API ** commons-lang3 v3.x Jenkins API <b>Timestamp</b> ** Caffeine API ** Script Security ** Ionicons API ** JAXB  ** - dépendance requise
✓ Timestamp	Workspace Cleanup	Ant	Gradle	
Pipeline	GitHub Branch Source	Pipeline: GitHub Groovy Libraries	Pipeline: Stage View	
Git	SSH Build Agents	Matrix Authorization Strategy	PAM Authentication	
LDAP	Email Extension	Mailer		

Jenkins 2.361.4

Fig: Installation des plugins de base.

# Créer le 1er utilisateur Administrateur

Nom d'utilisateur:

Mot de passe:

Confirmation du mot de passe:

Nom complet:

Adresse courriel:

Fig: Création d'un premier user

# Présentation de l'interface Jenkins

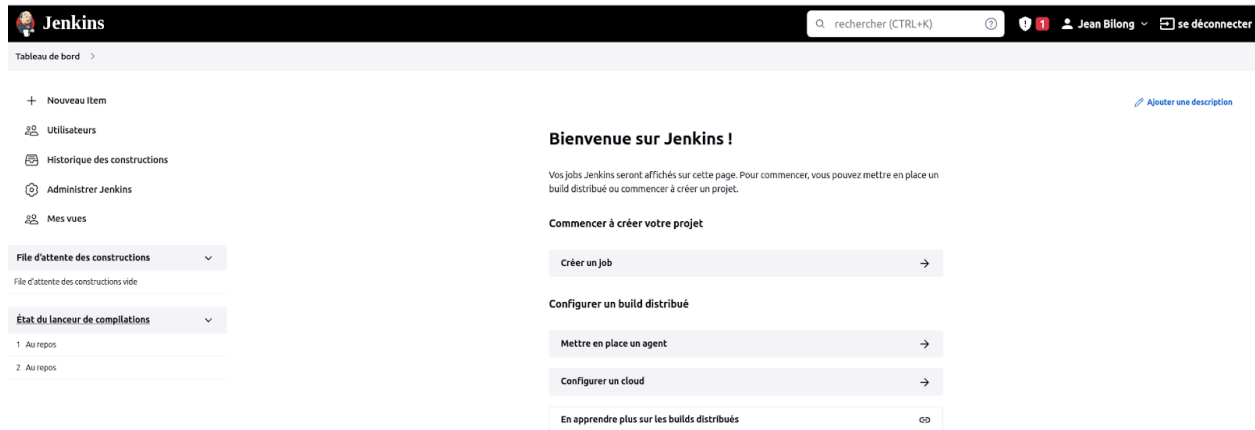


Fig : interface d'accueil de jenkins


L'interface se compose de plusieurs sections :


- **Le Header** est constitué du bandeau noir et du bandeau de file d'ariane pour pister la navigation. Il permet de signaler des informations importante comme des alertes sécurités sur les plugins à mettre à jours
- **Le Footer** où l'on retrouve le lien vers les api de jenkins ainsi que la version installés
- **Le corps de la page** avec une section de gauche donnant des informations sur le statut des jobs et des workers (actuellement au repos car aucun jobs ne tournent), un historique d'exécution des jobs, les différentes vues de l'utilisateur et l'accès à l'administration de Jenkins. Alors la section de gauche permet de lancer rapidement la création de nouveau jobs, de créer des jobs ou pipeline distribués soit en définissant un agent (worker) ou une ressource cloud.


# Création de notre premier job en mode free style


La création d'un job peut se faire soit en cliquant sur "Nouveau Item" dans la section de gauche ou "Créer un job" dans la section de droite.


**Saisissez un nom**  
  
» Champ obligatoire

**Construire un projet free-style**  
Ceci est la fonction principale de Jenkins qui sert à builder (construire) votre projet. Vous pouvez intégrer tous les outils de gestion de version avec tous les systèmes de build. Il est même possible d'utiliser Jenkins pour tout autre chose qu'un build logiciel.

**Pipeline**  
Organise des activités de longue durée qui peuvent s'étendre sur plusieurs agents de construction. Adapté pour la création des pipelines (anciennement connues comme workflows) et/ou pour organiser des activités complexes qui ne s'adaptent pas facilement à des tâches de type libre.

**Construire un projet multi-configuration**  
Adapté aux projets qui nécessitent un grand nombre de configurations différentes, comme des environnements de test multiples, des binaires spécifiques à une plateforme, etc.

**Dossier**  
Crée un conteneur qui stocke des objets imbriqués. Utile pour grouper ensemble des éléments. Contrairement à une vue qui n'est qu'un filtre, un dossier crée un espace de nommage distinct, de sorte que vous pouvez avoir plusieurs éléments du même nom tant qu'ils se trouvent dans des dossiers différents.

**Organization Folder**  
Creates a set of multibranch project subfolders by scanning for repositories.


**Pipeline Multibranches**  
Crée un ensemble de projets Pipeline en se basant sur les branches détectées dans le dépôt d'un gestionnaire de code source.

Fig: Choix du type de job à créer (freestyle)

Tableau de bord > MonPremierJob >

### Configuration

- General
- Gestion de code source
- Ce qui déclenche le build
- Environnements de Build
- Build Steps
- Actions à la suite du build

### General

Enabled ☒

Description

Ceci est mon premier job

[Plain text] [Prévisualisation](#)

☐ Ce build a des paramètres ?

☐ GitHub project

☐ Supprimer les anciens builds ?

☐ Throttle builds ?

☐ Exécuter des builds simultanément si nécessaire ?

[Avancé...](#)

### Gestion de code source

☒ Aucune

☐ Git ?

### Ce qui déclenche le build

[Sauver](#) [Apply](#)

Fig: Définitions des paramètres du job

Dans cette page on a 6 sections pour définir notre job:

**La section générale:** permet de déclarer la configuration générale du job, par exemple donner une description, déclarer des paramètres, indiquer si le projet que l'on build est un projet github etc.

**La gestion du code source** quant à elle permet de déclarer si on utilise git ou non.

**La section déclenchement du job** sert à définir si l'on souhaite déclencher un job automatique à distance en offrant plusieurs possibilités; par exemple par les triggers, par scrutation du code source, par le déclenchement en cascade suite à l'exécution des jobs précédents ou par définition de la période de déclenchement.

**La section environnement de build** permet de gérer les ressources de travail du build, par exemple le nettoyage de l'espace de travail avant le démarrage du job.



**La section Build step** permet d'ajouter des steps au Job c'est-à-dire les actions ou tâches que doit réaliser le build en exécution par exemple un script shell ou windows.

**La section Action à la suite du build** permet de définir des actions à réaliser à la fin d'exécution du job tels que, la notification par email, la publication d'un tag sur un dépôt git, l'archivage des artefacts ou le nettoyage du workspace, la publication d'un rapport de tests etc.

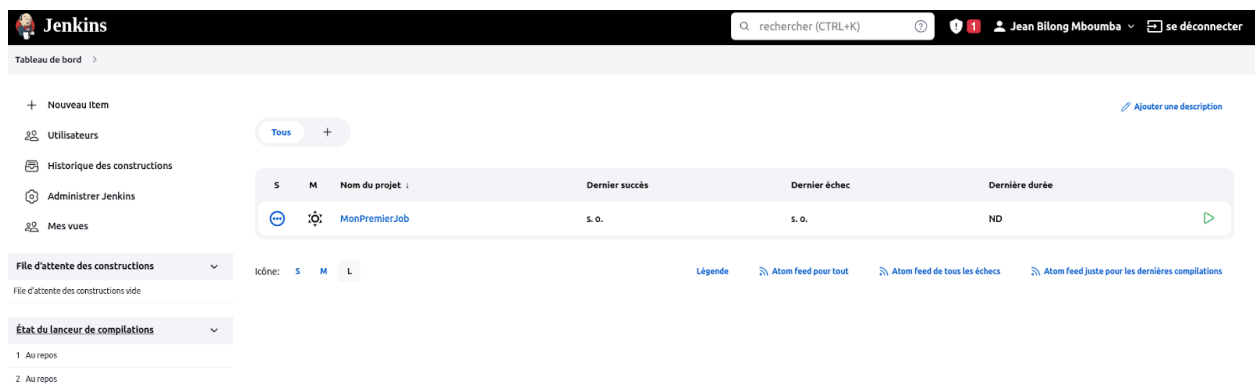


Fig: dashboard listant notre premier jobs

## Exécution de mon premier job

Pour exécuter notre premier job manuellement il suffit de cliquer sur le bouton lancer un job.

L'historique des exécutions de notre job est rempli à chaque clique. On peut voir des icônes vertes en cas de succès et rouge si le job s'est déclenchée avec des erreurs.

[↑ Retour au tableau de bord](#)

**Projet MonPremierJob**

↑ État

</> Modifications

📁 Répertoire de travail

▶ Lancer un build

⚙️ Configurer

🗑️ Supprimer Projet

✎ Renommer

☁ Historique des builds tendance ▾

🔍 Filter builds...

✓ #6 22 nov. 2022 07:19

✗ #5 22 nov. 2022 07:19

✗ #4 22 nov. 2022 07:19

✓ #3 22 nov. 2022 07:13

✓ #2 22 nov. 2022 07:13

✓ #1 22 nov. 2022 07:13

📡 [Atom feed des builds](#)

📡 [Atom feed des échecs](#)

**Liens permanents**

- [Dernier build \(#5\), il y a 17 s](#)
- [Dernier build stable \(#3\), il y a 5 mn 45 s](#)
- [Dernier build avec succès \(#3\), il y a 5 mn 45 s](#)
- [Dernier build en échec \(#5\), il y a 17 s](#)
- [Dernier build non réussi \(#5\), il y a 17 s](#)
- [Last completed build \(#5\), il y a 17 s](#)

Fig : Historique d'exécution de notre job

Pour voir le résultat d'exécution de notre job, il suffit de cliquer sur un item dans l'historique des builds.

Les deux figures ci-dessous montrent le détail d'exécution du job en en succès et en en échec.

Tableau de bord > MonPremierJob > #6

↑ Retour au projet

📄 État

</> Modifications

📄 **Sortie de la console**

📄 Voir en texte brut

☑ Informations de la construction

🗑 Supprimer le build "#6"

← Build précédent

✓ **Sortie de la console**

Started by user [Jean Bilong Mboumba](#)  
Running as SYSTEM  
Building in workspace /var/jenkins\_home/workspace/MonPremierJob  
[MonPremierJob] \$ /bin/sh -xe /tmp/jenkins16299457631609173196.sh  
+ echo Hello World  
Hello World  
Finished: SUCCESS

Fig: Succès d'exécution du build

Tableau de bord > MonPremierJob > #5

↑ Retour au projet

📄 État

</> Modifications

📄 **Sortie de la console**

📄 Voir en texte brut

☑ Informations de la construction

🗑 Supprimer le build "#5"

← Build précédent

→ Build suivant

✗ **Sortie de la console**

Started by user [Jean Bilong Mboumba](#)  
Running as SYSTEM  
Building in workspace /var/jenkins\_home/workspace/MonPremierJob  
[MonPremierJob] \$ /bin/sh -xe /tmp/jenkins4697873627868697277.sh  
+ echffo Hello World  
/tmp/jenkins4697873627868697277.sh: 2: echffo: not found  
Build step 'Execute shell' marked build as failure  
Finished: FAILURE

Fig: Échec d'exécution du build





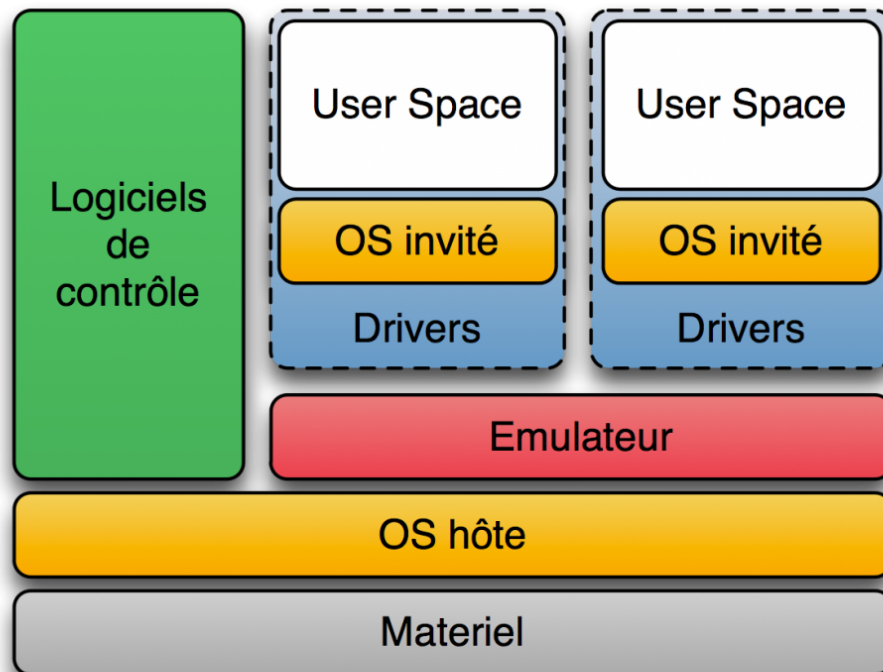


# Annexe 1 : Introduction à docker

## Définition

Docker est un **logiciel libre** permettant de lancer des **applications** dans **des conteneurs logiciels**.

C'est donc est un outil qui peut **empaqueter une application et ses dépendances** dans un **conteneur isolé**, qui pourra être exécuté sur n'importe quel serveur.



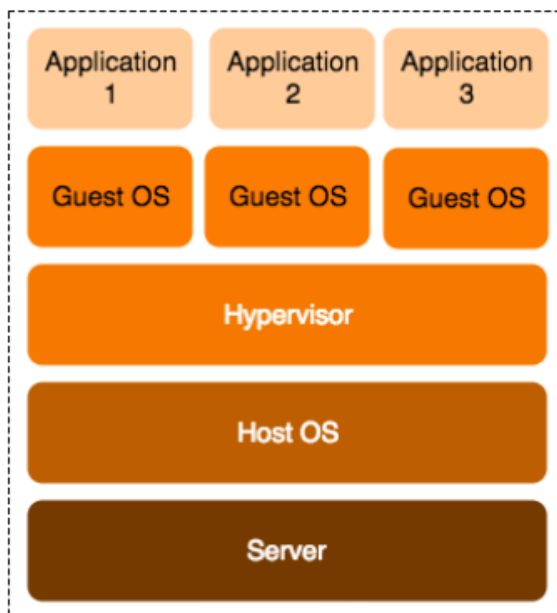
Il ne s'agit pas de **virtualisation**, mais de **conteneurisation**, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement.

Cette approche permet d'accroître la **flexibilité** et la **portabilité d'exécution d'une application**, laquelle va pouvoir tourner de façon **fiable** et **prévisible** sur **une grande variété de machines hôtes**, que ce soit sur la machine **locale**, un **cloud privé** ou **public**, une **machine nue**, etc

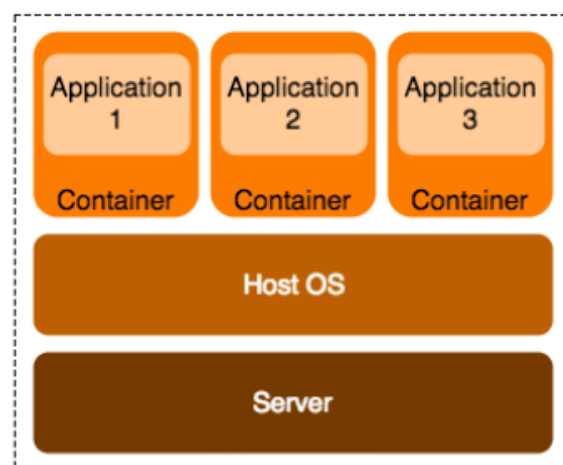
Plus généralement, lorsqu'on parle de **conteneur**, on parle de **virtualisation d'OS** qui s'appuie sur le standard de conteneur **linux LXC** (contraction de l'anglais **Linux Containers** est un **système de virtualisation**, utilisant **l'isolation** comme méthode de cloisonnement au niveau du **système d'exploitation**).

Il est utilisé pour faire fonctionner **des environnements Linux isolés les uns des autres** dans **des conteneurs**, **partageant le même noyau** et une plus ou moins grande partie du système hôte.

Le conteneur apporte une **virtualisation de l'environnement d'exécution** (processeur, mémoire vive, réseau, système de fichier...) et non pas de la machine.



(a) KVM

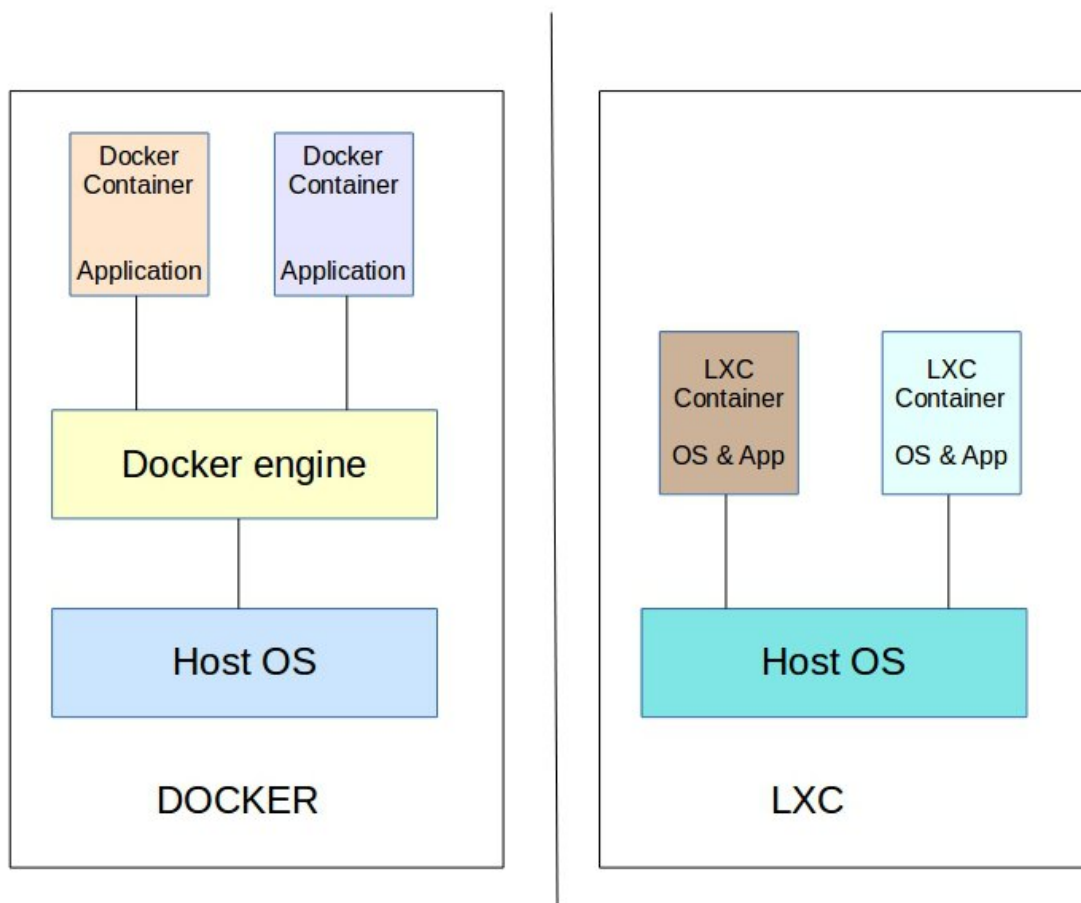


(b) LXC

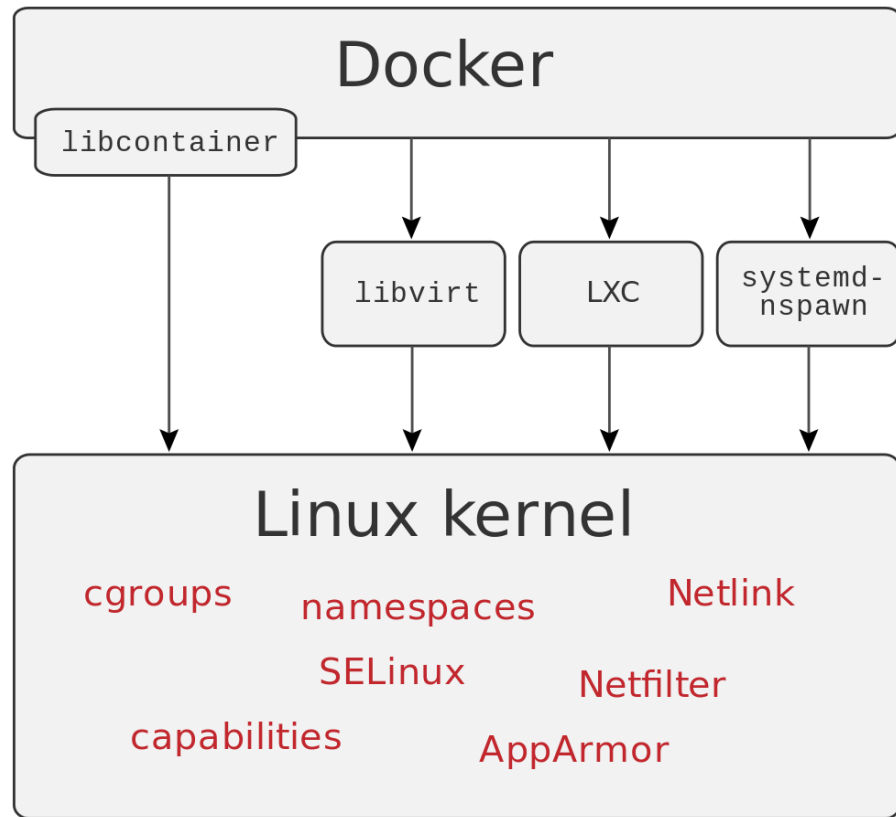


Pour cette raison, on parle de « **conteneur** » et non de « **machine virtuelle** ». au niveau du **système d'exploitation**: contrairement à la **virtualisation** qui **émule par logiciel** différentes **machines** sur une machine physique, **la conteneurisation émule différents OS sur un seul OS**.

Techniquement, Docker étend le format de conteneur **Linux standard, LXC**, avec une **API** de haut niveau fournissant une solution pratique de virtualisation qui exécute les processus de **façon isolée**.



Les Conteneurs docker sont basés sur **Alpine Linux** qui est une **distribution Linux ultra-légère, orientée sécurité et basée sur Musl** (en) et **BusyBox** (est un logiciel qui implémente un grand nombre des commandes standard sous Unix, à l'instar des GNU Core Utilities).



### *Qu'est-ce qu'une image Docker?*

Une image Docker est un fichier immuable, qui constitue une capture instantanée d'un conteneur.

Généralement, les images sont créées avec la commande « `docker build` ». Et puis, ils vont produire un conteneur quand ils sont lancés avec la commande « `run` ».

En revanche, dans un registre Docker, les images sont stockées comme « `registry.hub.docker.com` ».

Comme elles peuvent devenir assez volumineuses, les images sont conçues pour composer des couches de d'autres images, ce qui permet

d'envoyer une quantité minimale de données lors du transfert des images sur le réseau.

Les images sont stockées dans **un registre Docker**, tel que **Docker Hub**, et peuvent être téléchargées à l'aide de la commande **docker pull**.

### *Qu'est-ce qu'un conteneur Docker?*

Un conteneur Docker est une instance exécutable d'une image. En utilisant l'API ou la CLI de Docker, nous pouvons **créer**, **démarrer**, **arrêter**, **déplacer** ou **supprimer** un conteneur.

De manière avantageuse, nous pouvons connecter un **conteneur** à un ou plusieurs **réseaux**, y attacher de la **mémoire** ou créer une nouvelle image **sur la base de son état actuel**.

Si on applique le concept de **l'orienté objet**. **Si une image est une classe, un conteneur est une instance d'une classe**, c'est-à-dire un objet **d'exécution**

### *Conclusion*

Ce que vous devez retenir, c'est que les images sont des instantanés figés de conteneurs vivants. Alors que les conteneurs exécutent les instances d'une image.

<https://www.youtube.com/watch?v=caXHwYC3tq8>

### **Les commandes de base à connaître**

***docker pull <nom-image>*** : téléchargement d'une image docker depuis le repo <https://hub.docker.com/>; c'est la version latest qui est téléchargée

***docker pull <nom-image:versionId>*** : Récupère une version spécifique

***docker images*** : liste les images télécharger sur son pc

***docker system prune -a*** : supprime toutes les images docker

***docker run -d <nom-image>***: création d'une instance docker à partir de l'image

- Télécharge l'image si elle n'est sur le disque
- L'option <-d> exécute l'instance en daemon

***docker run -d --name <nom-local> <nom-image>***: création d'une instance docker à partir de l'image avec attribution d'un nom local.

***docker ps***: affiche l'id des instances docker en cours d'exécution

***docker rm --force \$(docker ps -a -q)***: supprime toutes les instances docker en cours d'exécution

***docker exec -it <instance-name>/<instance-id> <shell>***: permet de se loguer dans l'instance docker à partir de l'id ou du nom de l'instance et sur un shell quelconque

***docker stop <instance-name>/<instance-id>*** : permet de se loguer dans l'instance docker à partir de l'id ou du nom de l'instance et sur un shell quelconque.

***docker-compose -f <stack.yml> up -d***: exécution multi conteneurs docker à partir d'un fichier YML

***Exemples:***

1. [https://hub.docker.com/\\_/ghost](https://hub.docker.com/_/ghost)
2. <https://github.com/ynovmaster2/docker-intro>

## Annexe 2: Ghost (moteur de blog)

Ghost est un moteur de blog libre et open source écrit en JavaScript (NodeJs) et distribué sous licence MIT. Ghost est conçu pour simplifier le processus de publication en ligne par des blogueurs. (wikipedia)

# by default, the Ghost image will use SQLite (and thus requires no separate database container)

# we have used MySQL here merely for demonstration purposes (especially environment-variable-based configuration)

version: '3.1'

services:

ghost:

image: ghost:3-alpine

restart: always

ports:

- 2368:2368

environment:

# see

<https://docs.ghost.org/docs/config#section-running-ghost-with-config-env-variables>

database\_\_client: mysql

database\_\_connection\_\_host: db

database\_\_connection\_\_user: root

database\_\_connection\_\_password: example

database\_\_connection\_\_database: ghost

# this url value is just an example, and is likely wrong for your environment!

url: http://localhost:2368

db:

image: mysql:5.7

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: example