

Table des matières

Introduction générale.....	1
Chapitre 1 : Contexte général du projet.....	2
1.1. Introduction	3
1.2. Présentation de l'entreprise :	3
1.2.1. Historique	3
1.2.2. Vision et mission :	3
1.2.3. Les valeurs.....	4
1.2.4. Les services	4
1.2.5. Les solutions.....	5
1.2.6. Les partenaires :	5
1.3. Cadre du projet	6
1.3.1. Etude de l'existant	6
1.3.2. Limite de l'existant.....	6
1.3.3. Solutions proposées	7
1.3.4. Architecture générale de la solution	7
1.4. Conclusion.....	11
Chapitre 2 : État de l'art	12
2.1. Introduction	13
2.2. Intégration continue.....	13
2.2.1. Définition de l'intégration continue	13
2.2.2. Fonctionnement de l'IC	13
2.2.3. Les étapes de l'Intégration Continue	14
2.2.4. Les différents composants d'une plate-forme d'IC:	15
2.2.5. Les avantages de l'intégration continue.....	15
2.3. La virtualisation.....	16
2.3.1. Principe.....	16
2.3.2. Intérêts	17
2.3.3. Catégories de virtualisation	17
2.3.4. Présentation de l'Hyperviseur	18
2.4. Solutions de virtualisation	19
2.5. Technologie Docker	20
2.6. Etude comparative :	21
2.6.1. Gestionnaire de version : Git vs SVN	21

2.6.2. Serveur d'orchestration : Jenkins vs Bamboo vs Travis.....	23
2.6.3. Serveur de gestion de dépôt maven : Nexus vs JFrog Artifactory.....	24
2.6.4. Sonarqube vs SonarLint	25
2.6.5. La différence entre les conteneurs et les machines virtuelles.....	25
2.7. Choix de solution de virtualisation :.....	26
2.8. Les scripts YAML et Groovy	26
2.9. Conclusion.....	27
Chapitre3 : Réalisation	28
3.1. Introduction	29
3.2. Réalisation technique et environnement.....	29
3.2.1. Environnement de travail	29
3.2.2. Première connexion au serveur :	30
3.2.3. Manipulation technique via docker et docker-compose :.....	34
3.2.4. Configuration avancé de jenkins :.....	38
3.3. Réalisation fonctionnelle.....	41
3.4. Conclusion :.....	49
Conclusion générale	50
Liste des Références.....	51
Annexe	52

Table des figures

Figure 1: Logo Yess Consulting	3
Figure 2: Les partenaires de Yess Consulting	5
Figure 3: Architecture existante	6
Figure 4: Logo SVN	7
Figure 5: Logo jenkins	8
Figure 6: Logo sonarqube.....	8
Figure 7: Logo nexus.....	9
Figure 8: Logo diagramme de composants	9
Figure 9: Diagramme de déploiement	10
Figure 10: Architecture à mettre en place	11
Figure 11: Les étapes de l'intégration continue	14
Figure 12: Fonctionnement de plusieurs machines virtuelles sur une machine physique	16
Figure 13: Hyperviseur de type 1	18
Figure 14: Hyperviseur de type 2	19
Figure 15: Différence entre Machine Virtuelle et Docker.....	26
Figure 16: Première connexion au serveur.....	30
Figure 17: Accès au serveur avec putty.....	30
Figure 18: Paramètre docker-compose pour jenkins	32
Figure 19: Paramètres docker-compose pour sonar	33
Figure 20: Paramètres docker-compose pour mysql	33
Figure 21: Paramètres docker-compose pour nexus.....	34
Figure 22: Répertoire du lancement de la plateforme	35
Figure 23: Lancement de la plateforme.....	35
Figure 24: Liste des images docker téléchargées	36
Figure 25: Suppression de la plateforme	36
Figure 26: Liste des composants par docker-compose	37
Figure 27: Liste des composants par docker engine.....	37
Figure 28: Installation du maven dans jenkins	37
Figure 29: Copie du mot de passe de verrouillage de jenkins	38
Figure 30: Création du premier utilisateur jenkins.....	38
Figure 31: Configuration jenkins-sonarqube.....	39
Figure 32: Interface jenkins pour soumettre un plugin.....	39
Figure 33: Création d'un utilisateur nexus sous jenkins	40
Figure 34: Configuration jenkins-nexus	40
Figure 35: Interface principale de jenkins	41
Figure 36: Création d'un nouveau projet jenkins.....	41
Figure 37: Script groovy pour le pipeline jenkins	42
Figure 38: Création d'un utilisateur SVN pour le téléchargement du projet	43
Figure 39: Téléchargement du projet YessClever depuis SVN.....	43
Figure 40: Log correspond au stage du téléchargement	44
Figure 41: Vérification de la bonne installation java	44
Figure 42: Affichage de la version java	44
Figure 43: Stage de la suppression de l'espace du travail.....	45
Figure 44: Sortie de la console pour le stage « clean »	45
Figure 45: Compilation du code source.....	46
Figure 46: Scan du projet avec sonarqube.....	46

Figure 47: Déroulement du stage SonarQube Scan.....	46
Figure 48: Interface sonarqube pour le projet YessClever	47
Figure 49: Génération du livrable.....	47
Figure 50: Envoi du livrable vers nexus.....	48
Figure 51: Livrable sous nexus.....	48

Liste des Tableaux

Tableau 1: Tableau comparatif Git vs SVN.....	22
Tableau 2: Tableau comparatif Jenkins vs Bamboo vs Travis.....	24
Tableau 3: Tableau comparatif JFrog Artifactory vs Nexus sonatype.....	24
Tableau 4: Tableau comparatif Sonarqube vs SonarLint.....	25
Tableau 5. Caractéristiques du serveur utilisé.....	29

Introduction générale

Habituellement la mise en production d'une application est l'étape ultime d'un processus bien élaboré faisant intervenir des équipes différentes, à savoir l'équipe de développement et celle des tests. Ainsi, le développement, le test et la mise en production sont considérés comme trois étapes distinctes.

Faire intervenir autant d'équipes peut mener à des conflits puisque le but de chaque équipe est différent de l'autre. Quand les développeurs souhaitent innover et faire évoluer les applications, l'équipe production cherche, avant tout, à maintenir la stabilité du système informatique. D'ailleurs, chacun suit ses processus, et travaille avec ses propres outils qui sont rarement communicants. En conséquence, les relations entre ces équipes peuvent être conflictuelles. Ces conflits génèrent des retards de livraison, des coûts supplémentaires pour l'entreprise qui n'ont pas été prévu au départ, avec un impact sur le client dont sa satisfaction est le centre des préoccupations de l'entreprise.

Il devient alors évident qu'il faut adopter une nouvelle approche qui permet d'unifier le processus de développement et de production afin d'éviter tous les problèmes cités précédemment.

De ce fait, la notion de DevOps est née. Il s'agit d'une approche qui se base sur la synergie entre la partie opérationnelle et la partie production. L'alignement de l'ensemble des équipes du système d'information sur un objectif commun permet de réduire les conflits entre ces différents intervenants, d'éviter les retards dus à la communication entre eux, et d'améliorer, par conséquent, les délais des livraisons (Time-To-Market). C'est dans ce cadre que s'inscrit notre Projet de Fin d'Etude au sein de la société Yess Consulting, il s'agit de mettre en place une plateforme DevOps pour l'intégration continue du produit Yess Consulting, YessClever.

Ce rapport est composé de 3 volets :

- Dans le premier volet, nous présentons l'entreprise accueillante et le cadre général du projet.
- Le deuxième volet est consacré pour la définition des concepts de base de notre projet.
- En fin, nous décrivons la réalisation de notre projet.

Chapitre 1 : Contexte général du projet

Chapitre 1 : Contexte général du projet

1.1. Introduction

Le présent chapitre a pour objectif de mettre le projet dans son cadre général à savoir l'entreprise accueillante. Par la suite, nous enchaînerons avec l'étude de l'existant puis nous proposons la solution adéquate pour faire face à l'existant ainsi qu'une idée sur l'architecture générale de la solution à mettre en place.

1.2. Présentation de l'entreprise :

1.2.1. Historique

Yess Consulting a été fondé en janvier 2014, il s'agit d'un éditeur d'ERP et de solution avancée de système d'information innovants, que ce soit en mode web, Cloud ou encore sur mobile (Smartphones et tablettes).



Figure 1: Logo Yess Consulting

1.2.2. Vision et mission :

Yess Consulting vise à être la meilleure entreprise au monde et la plus recherchée pour des solutions personnalisées, la production d'ERP, le développement d'applications mobiles de solutions architecturales de sable.

Elle cherche à fournir à ses clients des logiciels fiables et supérieurs à un prix abordable tout en faisant du processus de développement une expérience agréable pour les clients et les employés. ^[1]

1.2.3. Les valeurs

La réussite d'un projet repose principalement sur la qualité de l'équipe projet. La compétence des consultants, leurs qualités humaines et la synergie du groupe constituent les clés du succès. C'est la raison pour laquelle ils tiennent à ce que chaque collaborateur respecte et véhicule les valeurs de Yess Consulting :

- Promettre moins mais livrer plus.
- Fournir des estimations d'effort honnêtes et un horaire de travail
- Assurer la transparence grâce à une communication proactive
- Travailler avec des clients qui sont eux-mêmes réussis et justes
- Construire des relations basées sur la confiance et le respect mutuel
- Embaucher et promouvoir en fonction du mérite et de la performance
- Inciter les employés à faire ressortir leur meilleur
- Ne faites jamais de discrimination - toujours un employeur offrant des chances égales

1.2.4. Les services

Yess Consulting fournit à ses clients quatre services :

- **AMOA**

Regroupe des consultants opérationnels ayant une expérience significative dans la gestion et la mise en œuvre de projets. Yess Consulting fournit également à ses clients des outils et des méthodologies spécifiques pour assurer un projet utile, livré dans le budget et le temps alloué

- **Design**

Yess Consulting conçoit des sites Web et des applications mobiles faciles à utiliser et à mettre à niveau. Son but: aider les clients à concevoir des produits et des entreprises qui génèrent un bénéfice net

- **Développement**

Yess Consulting conçoit, développons et fournit des applications natives de haute qualité. Elle identifie et développe la meilleure solution pour les besoins techniques et les défis commerciaux les plus difficiles des entreprises, faciles à mettre à jour. Sa sélection de systèmes de gestion de contenu est robuste, sécurisée et conviviale

- **ERP**

ERP Développe l'entreprise de manière plus intelligente, plus rapide et plus fiable grâce à un ERP de meilleure pratique moderne qui inclut tous les besoins du client. SIRH, CRM, SI (Stock Inventory), Comptabilité

1.2.5. Les solutions

La mission de Yess Consulting est de soutenir les entreprises vers l'excellence en répondant aux besoins de l'analyse, la simplification et l'optimisation des processus à la planification et à la gestion de toutes les ressources: Humains, Produits, Clients, médias et Finance.

Le produit YessClever se concentre sur les besoins du client en termes de : gestion des ressources humaines, gestion de la production et des stocks, gestion des relations clients et gestion des affaires.

Le YessClever ERP est disponible en deux modes SaaS et sur permise assure:

- Gestion du temps et des activités
- Gestion de la relation client (CRM)
- Gestion des talents
- Gestion de la comptabilité financière (multi-sociétés, multi-devises, multi-langues)

1.2.6. Les partenaires :

- Epic consulting
- Telecontrol Detection System (TDS)
- Global Innovative Consulting
- Arts



Figure 2: Les partenaires de Yess Consulting

1.3. Cadre du projet

1.3.1. Etude de l'existant

Yess consulting suit une procédure classique dans la gestion de ses projets, en effet l'équipe DEV met l'accent sur le développement des fonctionnalités.

L'équipe passe par une partie conceptuelle du produit, puis développe le code, en faisant des tests en local, et enfin génère le livrable pour ses clients.

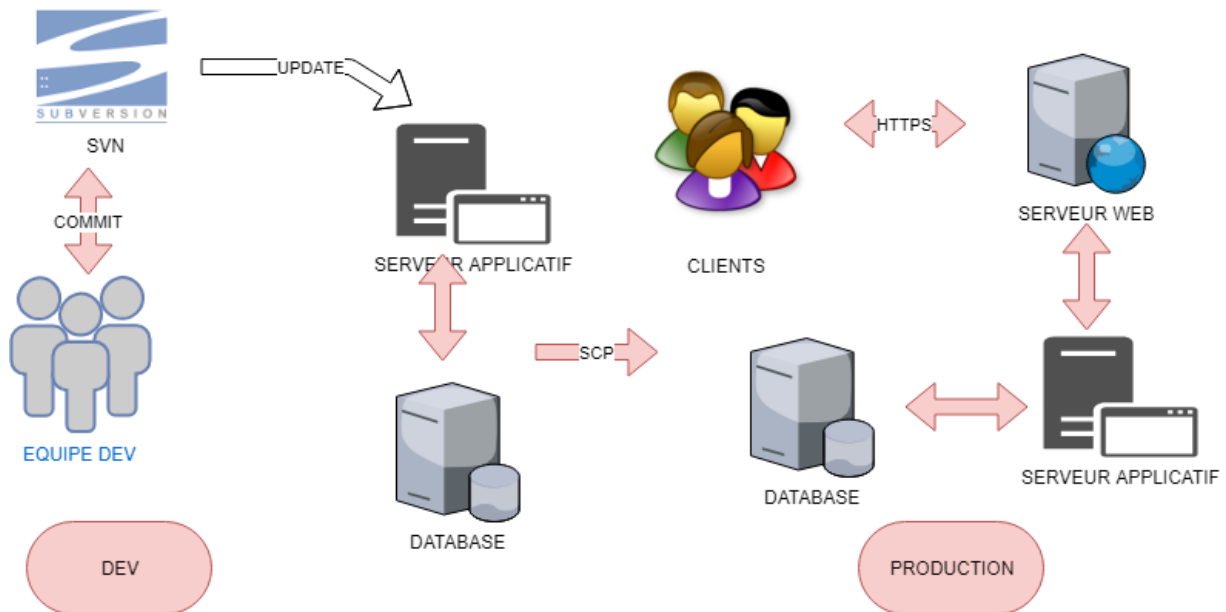


Figure 3: Architecture existante

Le produit tourne sur un serveur applicatif Tomcat avec une base de données postgresql, et un serveur web nginx.

1.3.2. Limite de l'existant

Après avoir étudié l'existant, nous pouvons extraire ses limites :

- Cycle de vie du produit incomplète avant le déploiement aux clients.
- Absence d'un serveur d'intégration continue.
- Le produit se livre sans exécuter les tests nécessaires tels que les tests unitaires et les mesures de la qualité de code.
- Les artefacts de l'application passent du local de l'équipe DEV au client directement dans l'absence du gestionnaire de dépôt.

Ça reste toujours des lacunes devant les ambitions d'une jeune entreprise qui vise un grand marché non seulement sur l'Afrique mais aussi sur l'Europe

1.3.3. Solution proposée

Pour faire face aux limites mentionnées ci-dessus, Yess Consulting nous a confié la tâche de l'étude et la mise en place d'une plateforme d'intégration continue.

Le pipeline d'intégration continu englobe la mise en place d'un serveur d'orchestration "Jenkins" et d'un serveur de gestionnaire de dépôt maven "Nexus sonatype" et d'un serveur de qualité de code "SonarQube".

Le moteur d'intégration continue jenkins, va être lié au serveur de gestion de version "SVN".

1.3.4. Architecture générale de la solution

➤ Composants de l'architecture

La plateforme que nous allons mettre en place est composée d'un serveur gestionnaire de source : SVN, un orchestrateur d'intégration continue : Jenkins, un serveur de qualité de code : Sonar et un serveur gestionnaire de dépôt : Nexus.

- **SVN**

Ce serveur est mis en place par l'entreprise accueillante, nous l'avons branché avec notre solution proposée pour gérer le cycle de vie du produit YessClever.

Les développeurs ont mis le code source du projet sous ce serveur, à chaque fois qu'il y'a un nouveau code, le développeur concerné met à jour le code sur le serveur SVN.

Le serveur SVN est lié directement avec le moteur de l'intégration continue Jenkins.



Figure 4: Logo SVN

- **Jenkins**

L'intégration continue se repose sur ce serveur, il gère tous les composants de l'architecture mise en place. Il est flexible ce qui permet de l'adapter avec plusieurs outils.

Nous choisissons Jenkins entre plusieurs autres outils, car il est le leader du marché de l'intégration continue, c'est le plus utilisé, et il offre plusieurs fonctionnalités permettant de bien gérer les applications.

Dans notre architecture, Jenkins télécharge le code source après chaque modification, détectée sous le serveur SVN, puis il s'occupe de terminer le cycle de vie de l'application.



Figure 5: Logo jenkins

- **Sonar**

Sonar est choisi comme serveur de qualité, il analyse code source et puis génère un rapport détaillé contenant plusieurs informations : nombre de lignes de code, pourcentage de couverture de code par les tests unitaires et nombre de violations de règles de codage.

Les tests réalisés par Sonar sont lancés par Jenkins automatiquement à chaque nouveau cycle de vie.

Sonar peut mettre les statuts du cycle en état « FAILURE » si la couverture du code n'atteint pas le pourcentage spécifié.

Sonar est lié avec un serveur de base de données MySQL.



Figure 6: Logo sonarqube

- **Nexus**

Cet outil sert à conserver les différentes versions générées par Jenkins après chaque lancement en état « SUCCESS », qui sont par la suite téléchargées et exploitées par les autres environnements tels que l'environnement QA et l'environnement production.

Jenkins peut automatiser le déploiement des artefacts sauvegardés dans Nexus, sous des serveurs applicatifs qui sont déployés dans des containers docker légers.



Figure 7: Logo nexus

- **Les principaux diagrammes de conception**

- **Diagramme de composants**

A travers ce diagramme, nous présentons l'ensemble des composants qui forme notre plateforme, ainsi que l'enchaînement des actions entre eux.

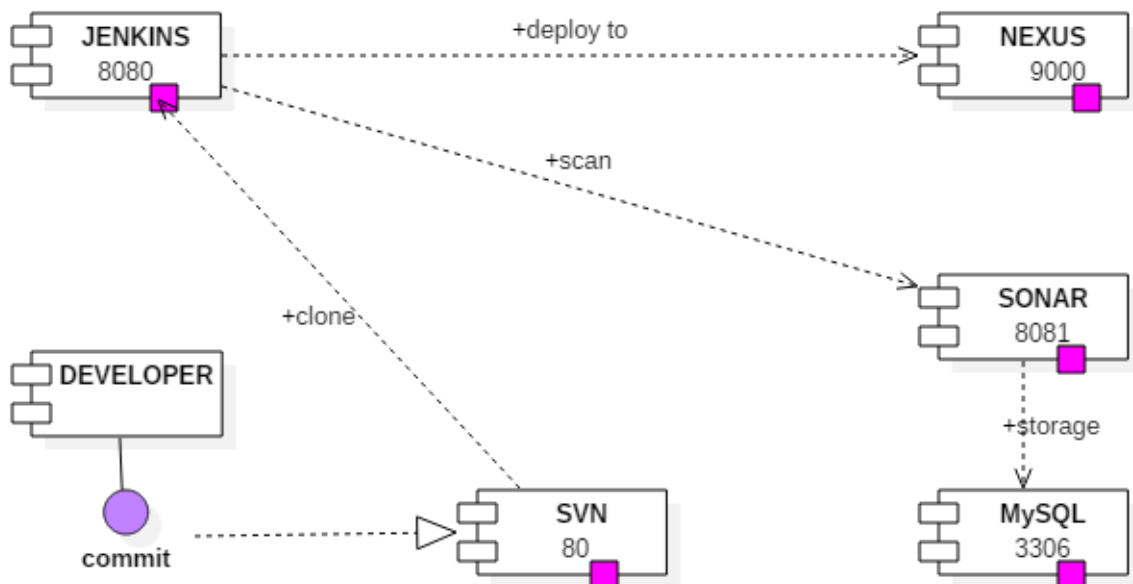


Figure 8: Logo diagramme de composants

Ce diagramme illustre l'enchaînement des actions les plus importantes réalisées par notre pipeline, en effet chaque développeur travaille dans son local, pour chaque nouvelle modification du code, il fait un commit vers le gestionnaire du code source.

Lorsque nous lançons un Build sous Jenkins, ce dernier télécharge le code source depuis SVN, le compile puis le scanne avec Sonarqube et enfin envoi la livrable générée vers le serveur de dépôt Nexus.

- **Diagramme de déploiement**

A travers ce diagramme, nous expliquons comment ce fait le déploiement de notre solution proposée :

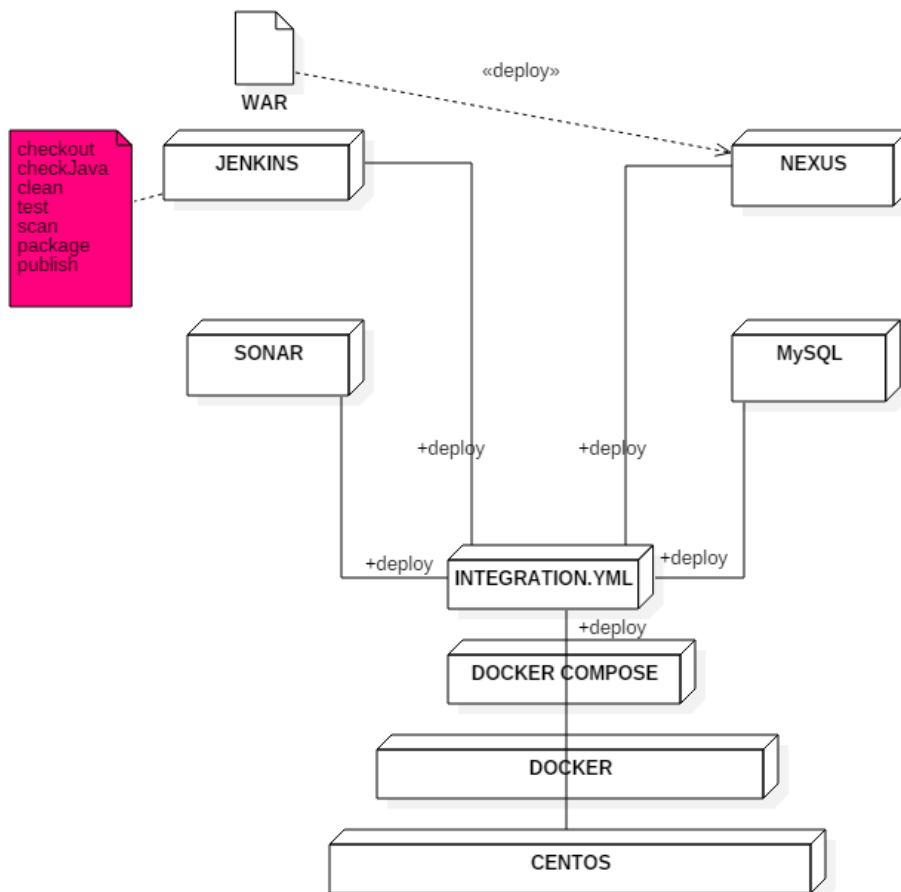


Figure 9: Diagramme de déploiement

Le diagramme de la figure 9, illustre toutes les étapes de déploiement de la plateforme d'intégration continue.

Il montre l'architecture complète qui assure le déploiement technique et fonctionnel de notre solution, dans le serveur hôte Centos, nous installons la solution de virtualisation docker, après nous installons docker-compose qui va assurer le lancement d'un script yaml pour mettre en place tous les composant : jenkins, sonar, mysql et nexus.

➤ Architecture à mettre en place

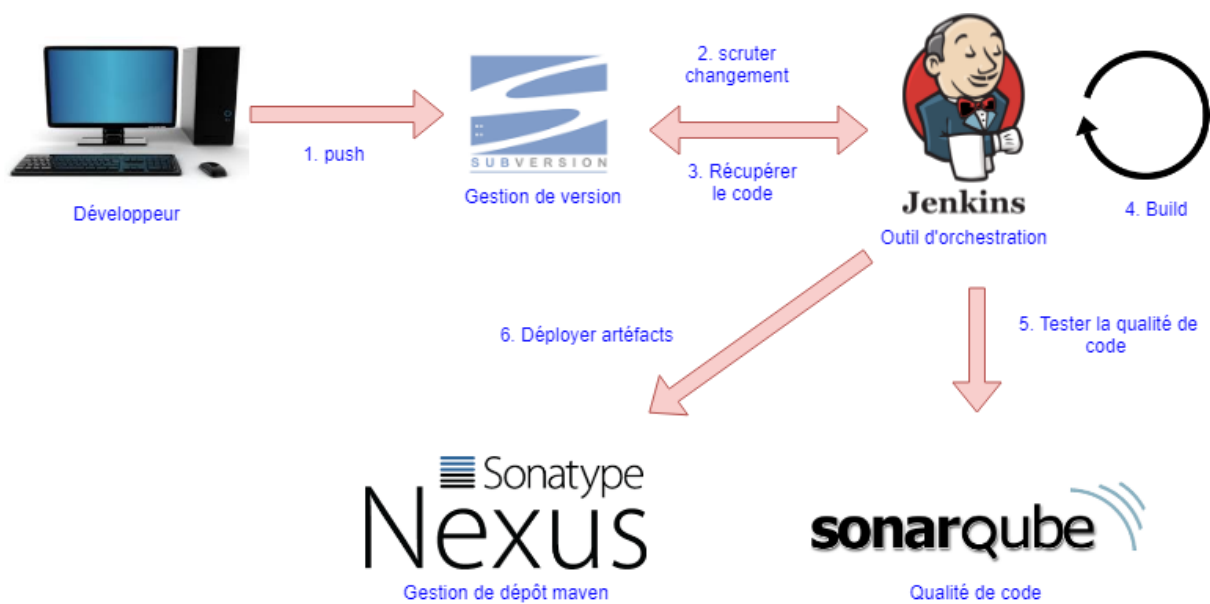


Figure 10: Architecture à mettre en place

Avec l'intégration continue nous souhaitons que les équipes R&D détectent les problèmes le plus tôt possible, c'est-à-dire, dès que le code soit écrit et non pas à quelques jours de la livraison.

1.4. Conclusion

Au cours de ce premier chapitre, nous avons commencé par présenter l'organisme d'accueil Yess Consulting. Par la suite nous avons étudié l'existant en citant ses limites puis nous avons expliqué la solution proposée en présentant son architecture. Nous détaillerons dans le chapitre suivant clairement la notion de l'intégration continue, la virtualisation traditionnelle et la migration vers la nouvelle technologie Docker.

Chapitre 2 : État de l'art

Chapitre 2 : Etat de l'art

2.1. Introduction

Nous commençons ce chapitre par la mise en évidence des différentes notions de base pour la réalisation de notre projet. Nous abordons le concept de l'intégration continue avec son fonctionnement et ses composants, puis nous expliquons la notion de virtualisation ainsi que sa nouvelle solution Docker.

2.2. Intégration continue

2.2.1. Définition de l'intégration continue

L'intégration continue est définie comme étant un ensemble de pratiques utilisées afin de s'assurer que chaque modification dans le code source ne produit pas des problèmes de régression dans l'application en cours de développement, c'est-à-dire qu'aucun défaut n'a été introduit à la partie du système qui n'a pas été modifié.

Le principal but de cette approche est d'anticiper et d'identifier rapidement les bugs avant la mise en production du logiciel. Ce qui permet d'avoir une vision plus complète du logiciel, notamment sur les différents points faibles et points forts du code ou de l'équipe. Ceci permet de gagner en réactivité pour faire face aux différents problèmes pouvant être présents dans les diverses phases du projet.

Pour bénéficier des avantages qu'apporte le concept d'intégration continue, il est nécessaire de suivre quelques règles de bonnes pratiques comme le maintien d'un unique dépôt pour le code source versionné, effectuer plusieurs commits par jour et par développeur, automatiser la compilation du code, et le maintien d'un délai de compilation court. ^[2]

2.2.2. Fonctionnement de l'IC

Cette méthode repose sur la mise en place d'une brique logicielle qui va permettre l'automatisation de plusieurs tâches :

- La compilation automatique du code et de ses dernières modifications,
- Les tests unitaires et fonctionnels,
- La validation du produit en fonction de plusieurs critères,
- Des tests de performances afin de procéder à certaines optimisations

Tout au long de l'évolution du projet, cette brique va exécuter un ensemble de tâches et de tests. Les résultats produits sont consultables par l'équipe de développeurs, pour comprendre ce qui pose problème dans les dernières modifications de code. Cette méthode d'intégration continue permet aussi de ne pas oublier certains éléments lors de la mise en production et donc d'améliorer la qualité de l'application.

2.2.3. Les étapes de l'Intégration Continue

- Le développeur va procéder à son commit : cela consiste en une opération de validation des dernières mises à jour du code source. Ces modifications sont effectuées en local, il faut donc exporter le code vers le serveur de gestion de source. Le développeur doit au passage assurer la bonne fusion de son code avec celui de ses voisins.
- L'ordonnanceur constate qu'une nouvelle version est disponible, il lance une compilation sur l'une des machines prévues à cet effet.
- Une compilation est effectuée sur une machine de compilation.
- Le code étant compilé, des tests unitaires sont lancés sur des machines de tests.
- En parallèle la qualité du code est vérifiée.
- Si la qualité du code et les tests unitaires sont satisfaisants, le(s) nouveau(x) binaire(s) est (sont) envoyé(s) dans le dépôt. Ils peuvent être déployés par la suite dans d'autres environnements.

En cas d'échec, une notification est générée au chef de projet et/ou à l'équipe de développement. Le développeur concerné par l'erreur fait un update du référentiel de gestion de configuration et corrige l'anomalie.

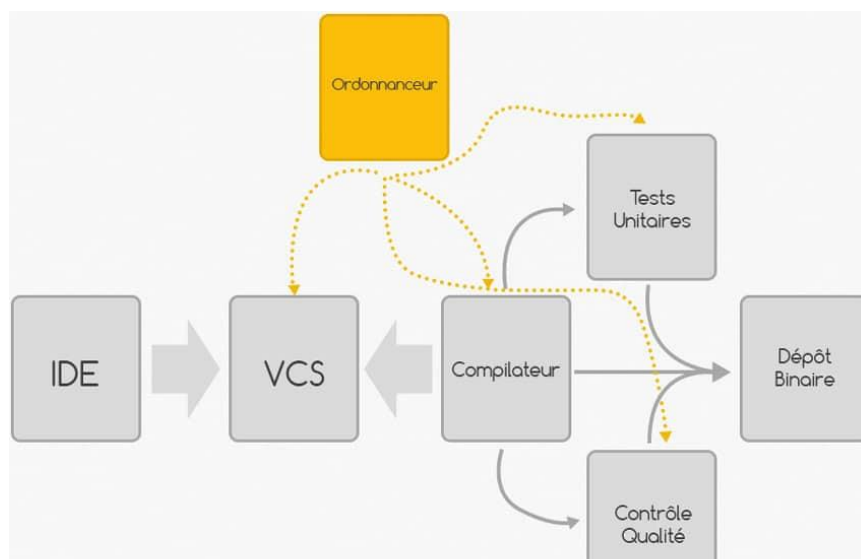


Figure 11: Les étapes de l'intégration continue

2.2.4. Les différents composants d'une plate-forme d'IC:

- L'ordonnanceur : c'est le chef d'orchestre du système. Il ordonne les tâches pour qu'elles se réalisent dans le bon ordre. Il remonte également les erreurs aux utilisateurs. L'ordonnanceur le plus classique à utiliser est Jenkins
- Le gestionnaire de source (VCS) : dans cette application se retrouve l'ensemble des sources sous différentes versions. Il est possible de revenir en arrière facilement, de créer des branches (pour différentes versions d'un même projet). Le gestionnaire de source classique est Git
- Le compilateur : c'est l'outil qui permet de traduire le code source (par exemple, du C) en langage compréhensible par la machine, ou par un interpréteur. Il en existe des milliers, les plus connus étant GCC et le JDK.
- Les tests unitaires : ce composant permet d'effectuer et de jouer l'ensemble des tests unitaires définis dans un projet.
- Le contrôle qualité : ce composant est chargé d'assurer la qualité du code en fonction de plusieurs paramètres. L'outil le plus classique est SonarQube
- Le dépôt de binaire : ce composant permet le stockage des différentes versions d'un projet après compilation. Il peut aussi servir comme source pour des composants utilisés dans son propre projet. L'outil Nexus peut servir comme moyen de dépôt binaire.

2.2.5. Les avantages de l'intégration continue

L'intégration continue a de nombreux avantages et qualités, tant en matière de développement, qu'en termes d'amélioration d'organisation de projet. Par exemple, la détection d'erreurs est faite plus rapidement tout en réduisant ainsi le délai nécessaire pour les corriger. Par conséquent, plus tôt une erreur est corrigée, moins la qualité du code final sera impactée. Outre que cet aspect, les bénéfices sont nombreux.

- Améliorer la productivité des développeurs qui passent moins de temps à corriger des bugs, ceux-ci sont repérés et corrigés plus rapidement, grâce à l'utilisation des tests unitaires.
- Amélioration de la visibilité sur le projet via un système de versions.
- Livrer plus rapidement des versions et des mises à jour.
- Meilleure intégration des applications aux environnements existants.

2.3. La virtualisation

La virtualisation c'est le fait de rendre logique une ressource technique ou physique dans le but d'optimiser l'utilisation des ressources et de réduire l'adhérence entre ces éléments. La virtualisation est une technologie de plus en plus incontournable, elle a été la première pierre vers l'ère du Cloud Computing. Bien que cette notion ne soit pas nouvelle, les évolutions quotidiennes au niveau des systèmes et des ressources mémoires offrent aujourd'hui un terrain fertile pour le développement rapide des solutions de virtualisation.

La virtualisation facilite la mutualisation des ressources. Les spécifications techniques des unités informatiques de traitement et de stockage du Cloud_Computing (Principe du cloud computing IaaS) sont transparentes pour l'utilisateur. La souplesse de montée en charge avec une capacité théoriquement infinie n'est pas le moindre des avantages. ^[3]

2.3.1. Principe

La virtualisation consiste à utiliser des moyens techniques (matériels et/ou logiciels) afin de faire fonctionner un ou plusieurs systèmes d'exploitation et/ou application sur un seul ordinateur comme s'ils fonctionnaient sur des ordinateurs distincts. Elle permet principalement une grande modularité dans la répartition des charges, ainsi une reconfiguration des serveurs en cas d'évolution ou de défaillance temporaire.

La figure 12 montre le principe de la virtualisation qui vise à faire fonctionner plusieurs machines virtuelles ayant chacune son propre système d'exploitation et qui partagent la même infrastructure physique.

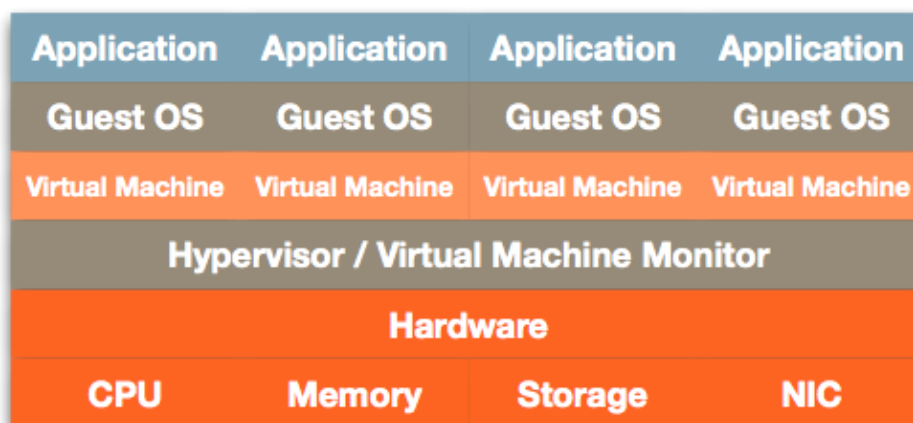


Figure 12: Fonctionnement de plusieurs machines virtuelles sur une machine physique

2.3.2. Intérêts

Les intérêts de la virtualisation sont nombreux nous citons principalement :

- Utilisation optimale des ressources d'un parc de machines (répartition des machines virtuelles sur les machines physiques en fonction des charges respectives).
- Installation, déploiement et migration facile des machines virtuelles d'une machine physique à une autre, notamment dans le contexte d'une mise en production à partir d'un environnement de qualification ou de pré-production, livraison facilitée.
- Economie sur le matériel par mutualisation (consommation électrique, entretien physique, surveillance, support, compatibilité matérielle, etc.).
- Sécurisation et/ou isolation d'un réseau (cassage des systèmes d'exploitation virtuels, mais pas des systèmes d'exploitation hôtes qui sont invisibles pour l'attaquant, tests d'architectures applicatives et réseau).

2.3.3. Catégories de virtualisation

Il existe différentes catégories de virtualisation.

➤ **Virtualisation du serveur**

La virtualisation du serveur permet de faire fonctionner et d'exécuter plusieurs machines virtuelles sur une seule machine comme s'ils fonctionnaient sur des machines physiques distinctes.

L'un des atouts le plus reconnu au niveau de cette catégorie de virtualisation, c'est qu'en cas de perte totale d'un centre d'hébergement ou en cas d'un événement exceptionnel il n'y aura pas un impact dramatique sur l'activité des machines.

➤ **Virtualisation du stockage**

De point de vue utilisateur, la virtualisation sert à masquer la disparité physique des unités de stockage et à le présenter comme un unique volume de données logique.

➤ **Virtualisation d'application**

La virtualisation permet de séparer l'application du système d'exploitation hôte et des autres applications présentes dans le but d'éviter les conflits. Cette solution est particulièrement consistante pour rendre moins compliqué l'administration des ressources informatiques notamment lors de l'installation des nouvelles versions.

➤ Virtualisation du poste client

Cette technologie impose la gestion des ressources du poste client par un serveur dans le Datacenter. (Quelque part un retour au client léger). La virtualisation du poste client est un moyen efficace pour maîtriser le coût de possession et simplifier la gestion et le déploiement, tout en assurant un contrôle constant et en temps réel pour une continuité d'activité optimale.

2.3.4. Présentation de l'Hyperviseur

Les technologies de virtualisation ajoutent une nouvelle couche à l'architecture classique au lieu du système d'exploitation. Cette couche est appelée Hyperviseur.

Un hyperviseur est une application qui prend en charge de la virtualisation logicielle d'un système d'exploitation. Il gère l'allocation de ressources matérielles réelles aux machines virtuelles hébergées.

Un hyperviseur est une plate-forme de virtualisation qui permet à plusieurs systèmes d'exploitation de travailler sur une même machine physique en même temps.

Il existe deux types d'hyperviseur : ^[4]

➤ Hyperviseur de type 1

Un hyperviseur de type 1 est un système qui s'installe directement sur la couche matérielle du serveur. Ces systèmes sont allégés de manière à se « concentrer » sur la gestion des systèmes d'exploitation invités c'est-à-dire ceux utilisés par les machines virtuelles qu'ils contiennent. Ceci permet de libérer le plus de ressources possible pour les machines virtuelles.

Parmi les hyperviseurs de type 1 nous trouvons des systèmes comme Xen, VMware ESX et Proxmox.

La figure suivante présente l'hyperviseur de type 1 :

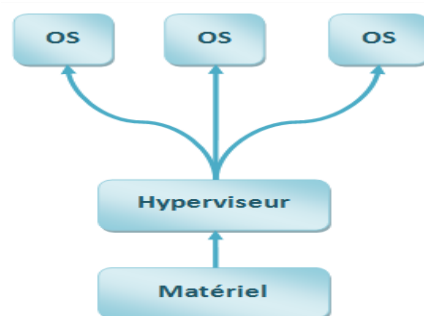


Figure 13: Hyperviseur de type 1

➤ Hyperviseur de type 2

Un hyperviseur de type 2 est un logiciel qui s'installe et s'exécute sur un système d'exploitation déjà en place. De ce fait, plus de ressources sont utilisées étant donné qu'on fait tourner l'hyperviseur et le système d'exploitation qui le supporte, il y a donc moins de ressources disponibles pour les machines virtuelles. L'intérêt qu'on peut trouver c'est le fait de pouvoir exécuter plusieurs hyperviseurs simultanément vu qu'ils ne sont pas liés à la couche matérielle.

Parmi les hyperviseurs de type 2, nous trouvons VMware Player, VMware Workstation, Virtual PC et Virtual Box.

La figure suivante présente l'hyperviseur de type 2 :

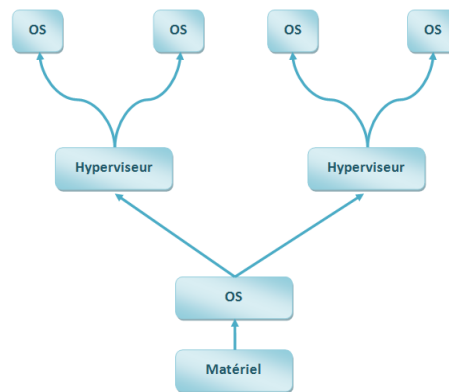


Figure 14: Hyperviseur de type 2

2.4. Solutions de virtualisation

Dans cette section, nous présentons les outils de virtualisation les plus utilisés qui sont : Xen, KVM, VMware et HyperV :

- **Xen** : est une solution libre de virtualisation permettant de faire tourner plusieurs systèmes d'exploitation sur une même machine physique. Il est de type hyperviseur, c'est à dire qu'il vient s'insérer entre le matériel et le noyau. Xen est considéré comme une solution à base de para virtualisation, car les systèmes invités doivent être modifiés pour cohabiter.

- **KVM** : est un projet de virtualisation complète qui est actuellement en développement pour un module de para virtualisation. Il est intégré depuis le noyau Linux 2.6.20 et permettant une virtualisation matérielle des processeurs.
- **VMware Server** : est une société qui offre des produits propriétaires liés à la virtualisation d'architectures x86. Elle est leader dans le marché de la virtualisation pour PC. Son produit de virtualisation VMware Server est de type virtualisation complète pour serveur sous GNU/Linux et/ou Microsoft Windows.
- **HyperV** : est une solution de virtualisation basée sur la virtualisation 64 bits pour Microsoft, Il est considéré comme une solution de para virtualisation.

Depuis l'année 2013, une nouvelle solution de virtualisation a été créée nommé « Docker », elle a été développée rapidement pour être un grand concurrent des autres solutions dans le marché de virtualisation surtout VMware.

2.5. Technologie Docker

➤ Définition

Docker est une solution de virtualisation légère pour Linux créée en 2013 et possédant une forte dynamique de développement et de déploiement.

Docker est une plateforme permettant de construire, déployer, et lancer des applications cloisonnées.

Contrairement à d'autres systèmes de virtualisation légère comme OpenVZ ou LXC qui ont des approches purement "virtualisation" d'un système, Docker se place dans un cadre applicatif.

L'objectif est de permettre aux développeurs et administrateurs de créer facilement des "Dockerfiles" décrivant un système : packages, fichiers de configuration, ports utilisés, etc.

Cette description permet ensuite à n'importe qui possédant Docker de construire le conteneur applicatif, assurant ainsi un environnement compatible.

La virtualisation légère de Docker se base sur les fonctionnalités fournies par le kernel Linux comme les cgroups et les kernel namespaces, permettant de garder des performances attendues d'un système de virtualisation légère. ^[5]

➤ Poids léger

Les conteneurs en cours d'exécution sur une seule machine partagent tous le même noyau du système d'exploitation afin qu'ils commencent instantanément et de faire une utilisation plus efficace de la RAM. Les images sont construites à partir de systèmes de fichiers en couches afin qu'ils puissent partager des fichiers communs, ce qui rend l'utilisation du disque et de l'image de téléchargements beaucoup plus efficace.

➤ Plateforme

La plateforme Docker est composée de deux éléments :

- Le démon Docker qui s'exécute en arrière-plan et qui s'occupe de gérer les conteneurs
- Le client Docker qui permet d'interagir avec le démon par l'intermédiaire d'un outil en ligne de commande

2.6. Etude comparative :

Pour mettre en place le pipeline d'intégration continue, nous avons besoin d'un ensemble d'outils à savoir, un gestionnaire de version, un orchestrateur, un serveur de dépôt maven et un serveur de qualité de code.

Dans cette partie, nous procédons à une étude comparative entre les différents outils qui existent sur le marché.

2.6.1. Gestionnaire de version : Git vs SVN

Git et SVN sont les deux outils de gestion de version les plus répandus sur le marché, ci-dessous un tableau comparatif entre ces deux outils ce qui permettra par la suite d'effectuer notre choix technique. ^[6]

Tableau 1: Tableau comparatif Git vs SVN

Aspect	SVN	Git
Architecture	Centralisé, uniquement le répertoire central a l'historique complet des changements. Les utilisateurs doivent communiquer via le réseau avec le répertoire central pour obtenir l'historique. Les backup sont gérés et maintenu indépendamment du SVN.	Distribué, tous les développeurs qui vérifient le code à partir d'un dépôt / serveur central auront leur propre référentiel cloné installé sur leur machine. Ce qui permet les développeurs de pouvoir créer une nouvelle branche, de faire un « commit » sur un fichier et de revoir une version même en absence de connexion.
Single point of Failure	Si le répertoire central est endommagé, alors uniquement les données enregistrées dans le dernier backup sont récupérable.	La notion de « single point of failure » n'existe pas car il y a autant de copie du dossier qu'il y'en a d'utilisateur si ce n'est plus.
Contrôle d'accès	Nécessité d'un « commit access » due au fait de la centralisation.	Non nécessite d'un « Commit access » puisque le système est distribué. Il faut juste décider de fusionner quoi à partir de quel utilisateur.
Stockage de contenu	Stockage des métadonnées des fichiers sous forme de dossier caché .svn	Stockage de tout le contenu dans un dossier .git, c'est le dépôt du code cloné sur la machine client.

Les branches	Les branches sont gérées comme étant un autre dossier dans le dépôt du code, elles ne sont pas historiées. Pour connaître s'il y a eu fusion entre des branches, il faut explicitement exécuter une commande. Cette façon de gérer les branches démultiplie les chances de créer des branches «orpheline». Une fois la fusion est faite nous ne pouvons plus savoir ce qui a été modifié.	La gestion des branches est assez simple, nous pouvons facilement naviguer entre les différentes branches d'un même répertoire de travail. Chaque répertoire de travail existant chez un utilisateur est considéré comme une branche. La fusion de la branche inclut les informations suivantes : auteur, heure, date, informations relatives à la branche et à la révision, les changements effectués par rapport à la version précédente. la fusion suivante, commence automatiquement là où la dernière fusion a été faite.
Espace	Le répertoire de travail contient deux copie de chaque fichier, un pour travailler avec et un caché dans le .svn, ce qui rend l'espace nécessaire relativement grand.	Taille du dépôt de code et répertoire de travail relativement petit (30X inférieur à celui de SVN). Ceci est due au fait que pour Git, le répertoire de travail nécessite seulement un fichier d'index de taille 100 bits pour

Cette comparaison a été faite pour un but académique, le choix du gestionnaire du code source a été fait par l'entreprise.

2.6.2. Serveur d'orchestration : Jenkins vs Bamboo vs Travis

Jenkins, travis et bamboo sont des outils d'orchestration bien connu sur le marché, chacun d'entre eux présente ses avantages et ses inconvénients. Le tableau 2 permet de comparer entre ces outils. La comparaison se base sur l'aspect Open source du code, le nombre de plugins, la simplicité d'utilisation et l'interfaçage avec les différents outils de gestion de version.

Tableau 2: Tableau comparatif Jenkins vs Bamboo vs Travis

Aspect	Jenkins	Bamboo	Travis
Open source	Oui	Non	Oui
Plugins	Oui	Cher	Peu
Simplicité	Non	Oui	Non
Outil de versionning	Tout	Tout	Github

L'étude comparative entre ces différents serveurs d'orchestration a dirigé notre choix vers Jenkins, qui, grâce à ses nombreux plugins, permet un interfaçage facile avec de nombreux outils dont nous avons besoin tel que Nexus et Sonar.

2.6.3. Serveur de gestion de dépôt maven : Nexus vs JFrog Artifactory

L'utilisation d'un gestionnaire de dépôt maven a plusieurs utilités, le dépôt peut servir d'un backup sur les applications dont les artefacts sont déployés sur ces serveurs, les artefacts peuvent être utilisés comme des bibliothèques et peuvent être importés par les différents développeurs sur d'autres applications.

Nexus et JFrog Artifactory sont les deux gestionnaires de dépôts maven les plus répandus sur le marché.

Ci-dessous nous présentons la comparaison entre ces deux outils, notre comparaison est basée sur :

- Le code (open source ou non)
- Installation à base de docker et dockerfile
- Taille du jar.

Tableau 3: Tableau comparatif JFrog Artifactory vs Nexus sonatype

Aspect	JFrog Artifactory	Nexus sonatype
Open source	Oui	Oui
Installation à base de Docker	Moyen	Facile
Dockerfile	Non disponible	Disponible
Taille du jar	37M	13M

2.6.4. Sonarqube vs SonarLint

Le serveur de qualité de code a plusieurs atouts pour n'importe quel projet, en effet, il fournit une analyse complète de la qualité d'une application en fournissant de nombreuses statistiques.

Ces données permettent ainsi d'évaluer la qualité du code, et d'en connaître l'évolution au cours du développement.

Ci-dessous nous présentons la comparaison entre ces deux outils, notre comparaison est basée sur :

- Le code (open source ou non)
- Installation à base de docker et dockerfile
- Langage supportés et l'existence ou non du tableau de bord. ^[7]

Tableau 4: Tableau comparatif Sonarqube vs SonarLint

Aspect	Sonarqube	Sonarlint
Open source	Oui	Oui
Installation à base de Docker	Facile	Moyen
Langages supportés	25+	20+
Tableau de bord	Oui	Non

Après cette comparaison, nous avons choisi de travailler avec Sonarqube comme serveur de qualité de code.

2.6.5. La différence entre les conteneurs et les machines virtuelles

Les conteneurs ont des avantages d'isolement des ressources et de répartition similaire à des machines virtuelles, mais une approche architecturale différente leur permet d'être beaucoup plus portables et efficaces.

Contrairement aux machines virtuelles traditionnelles, un conteneur Docker n'inclut pas de système d'exploitation, s'appuyant sur les fonctionnalités du système d'exploitation fournies par l'infrastructure sous-jacente.

- Machine virtuelle : Chaque machine virtuelle comprend l'application, les binaires et les bibliothèques nécessaires et tout un système d'exploitation invité - qui peuvent être des dizaines de GBs taille.
- Conteneur : Les conteneurs comprennent l'application et toutes ses dépendances, mais partagent le noyau avec d'autres conteneurs. Ils fonctionnent comme un processus isolé dans l'espace utilisateur sur le système d'exploitation hôte.

La figure 15 explique la différence entre la virtualisation traditionnelle par VMware et la nouvelle solution par Docker.

VMs vs Docker

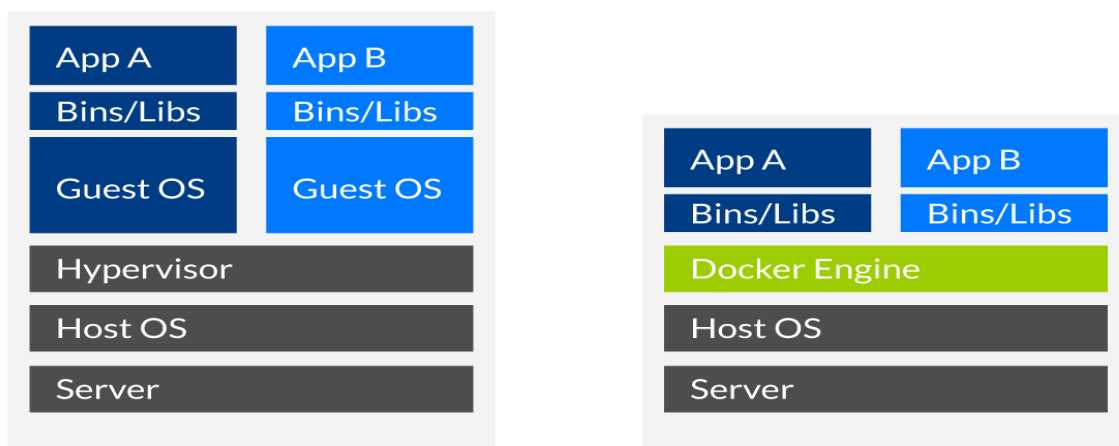


Figure 15: Différence entre Machine Virtuelle et Docker

2.7. Choix de solution de virtualisation :

Vu ses points forts, sa légèreté, son portabilité et l'isolation qu'il offre, nous avons décidé de choisir Docker comme solution de virtualisation pour mettre en place notre plateforme d'intégration continue.

2.8. Les scripts YAML et Groovy

Dans notre projet, nous avons utilisé deux types de scripts, YAML et Groovy

➤ YAML

Pour le développement des scripts, nous avons utilisé le langage de sérialisation YAML qui fournit de puissants paramètres de configuration, sans avoir à apprendre un type de code plus complexe comme CSS, JavaScript et PHP.

YAML est l'acronyme de «Yet Another Markup Language», il se définit comme étant « un standard de sérialisation de données pour tous les langages, facile à utiliser pour les humains ».

- Standard : car il n'y a pas d'approximation à avoir sur le contenu écrit en YAML.
- Sérialisation de données : consiste en la transformation d'une structure de données en un fichier texte.
- Human-friendly : une structure de données écrite en YAML est très simple à lire, facile à comprendre, et facile à éditer.

➤ **Groovy**

Groovy est le nom d'un langage de programmation orienté objet destiné à la plate-forme Java. Il constitue une alternative au langage Java pour cette plate-forme et est inspiré de Python, Ruby et Smalltalk.

Groovy utilise une syntaxe très proche de Java, avec des accolades, et est directement compilé, soit à la volée dynamiquement, soit classiquement avec un compilateur en bytecode.

Groovy s'intègre et est entièrement compatible avec la JVM étant donné que le bytecode est le même. Il peut donc :

- utiliser les bibliothèques Java ;
- être utilisé dans des classes Java.

Groovy peut être comparé à BeanShell, l'objectif de faire un langage de scripting proche de java est le même, la mise en œuvre étant différente.

2.9. Conclusion

Au cours de ce chapitre, nous avons présenté les concepts de base de notre projet, nous avons définis l'intégration continue, son fonctionnement et ses composants, la virtualisation avec ses solutions en se focalisant sur la technologie Docker, et enfin nous avons décrit nos choix concernant les outils de virtualisation ainsi que les types de scripts utilisés qui seront le noyau de la partie réalisation.

Chapitre3 : Réalisation

Chapitre 3 : Réalisation

3.1. Introduction

Dans le présent chapitre, nous allons présenter nos travaux techniques et fonctionnels pour mettre en place la plateforme d'intégration continue.




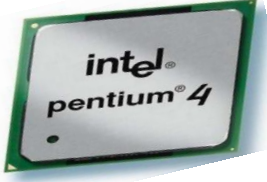

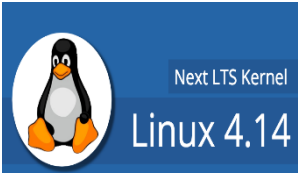
3.2. Réalisation technique et environnement

3.2.1. Environnement de travail

Pour mettre en place notre solution, nous avons choisi comme hébergeur cloud Novatis.

Les caractéristiques de notre serveur sont illustrées dans le tableau suivant.

Tableau 5. Caractéristiques du serveur utilisé

Serveur	DELL	
Type du système	64 bits	
Mémoire installé(RAM)	16 Go	
Processeur	4 CPUs	
Système d'exploitation	Centos Linux 7	
Noyau	Noyau 4.14.11	

3.2.2. Première connexion au serveur :

Il y'a beaucoup de manière pour se connecter au serveur, nous avons choisi la connexion SSH via Putty car il s'agit d'une connexion sécurisée :

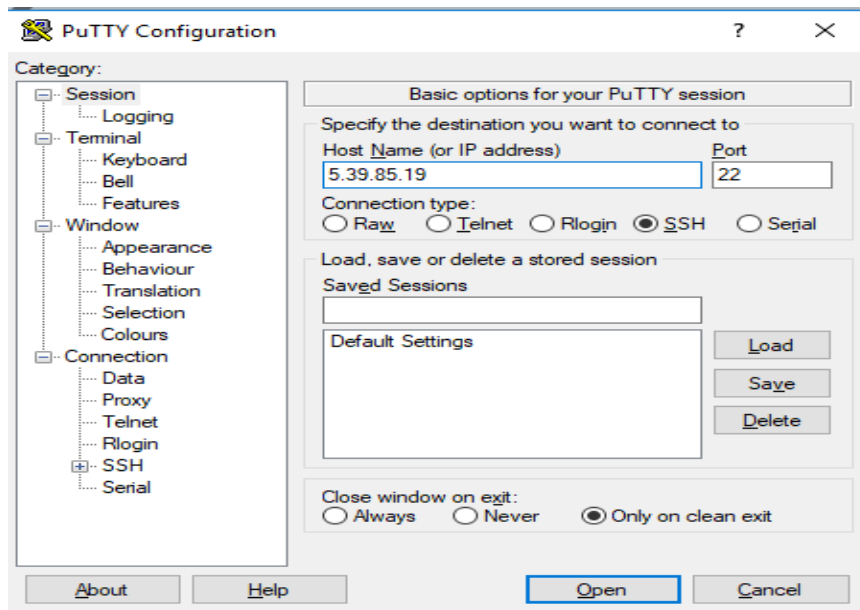


Figure 16: Première connexion au serveur

Après avoir cliqué sur open, la fenêtre suivante s'affiche :

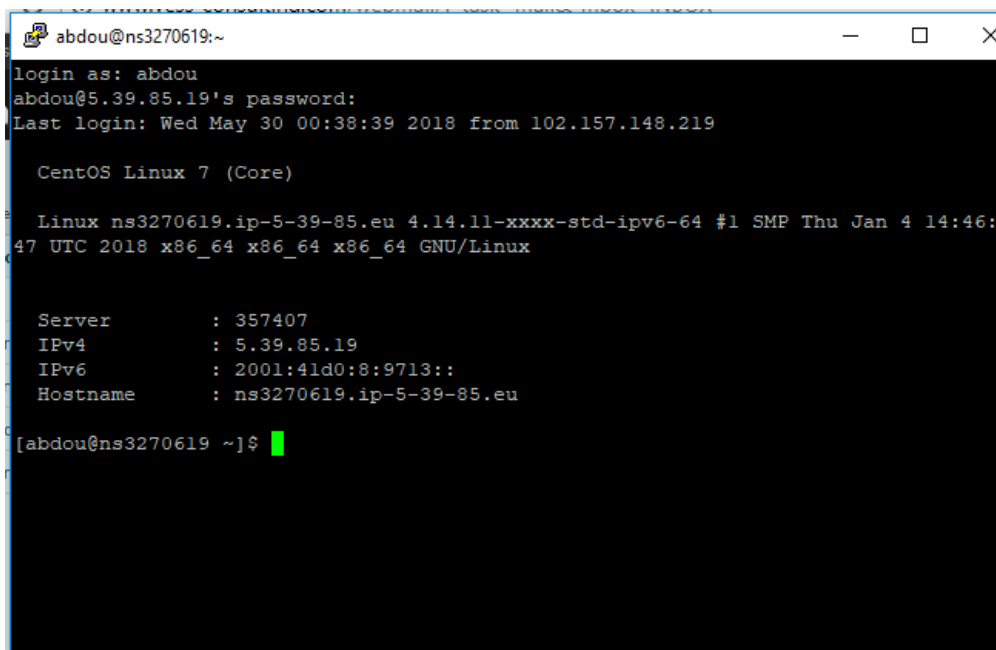


Figure 17: Accès au serveur avec putty

Après la connexion sur notre serveur, la première étape sera l'installation de docker et docker-compose (voir Annexe).

Docker va gérer les images et les conteneurs en ligne de commande, alors que docker-compose va les gérer en se basant sur un fichier yaml.

Génération du fichier yaml :

Comment nous l'avons expliqué, le fichier yaml englobe toute la configuration technique de notre plateforme,

Une fois le fichier exécuté par docker-compose, nous créons un environnement formé de quatre images : docker, jenkins, Nexus, Sonar et mysql.

Il est possible de créer un fichier séparé pour chaque composant de la plateforme, mais pour assurer la communication entre les composants (seul réseau), et pour faciliter le déploiement de la solution, nous décidons de mettre toute la configuration de notre plateforme dans un seul fichier yaml.

Notre yaml est de version 3, il est riche en nouvelles fonctionnalités et nous offre plusieurs caractéristiques.

Ci-dessous nous présenterons le contenu du fichier de configuration, nous avons nommé le fichier intégration.yml :

Le fichier est composé de quatre parties, chaque partie présente un composant de notre architecture à mettre en place.

Principe général

Pour déployer une application sous docker, il faut préciser l'image de base, à partir de laquelle s'instancie le container docker.

Chaque composant doit tourner sur un port précis, du coup il faut le noter dans la configuration,

Docker permet le mapping des ports entre le système d'exploitation et les containers, ce qui permet d'éviter les conflits des ports des applications qui tourne directement sur l'OS de base, et celle qui tournent sur docker.

Il faut partager les données importantes des containers docker avec l'OS, pour permettre la sauvegarde d'une copie de ces données au cas où le container crache.

Docker compose permet aussi de mettre les variables d'environnement dans le fichier yaml.

Nous pouvons aussi configurer les noms des composants de la plateforme.

Pour tourner la plateforme d'intégration continue, nous avons créé un fichier `intégration.yml` qui englobe tous les composants, avec toutes les configurations nécessaires.

En commençant à lire le fichier, les deux premières lignes précisent la version du yml utilisé ainsi que le mot « services » qui est commun pour tous les composants.

Nous pouvons déviser notre configuration en quatre bloc selon le nombre des services :

➤ Jenkins

```
jenkins:
  image: jenkins
  container_name: jenkins
  ports:
    - 10081:8080
    - 50000:50000
  volumes:
    - ./jenkins:/var/jenkins_home
    - /var/run/docker.sock:/var/run/docker.sock
  restart: unless-stopped
```

Figure 18: Paramètre docker-compose pour jenkins

Le container jenkins est instancié de l'image officielle jenkins qui se trouve sur le docker-hub, il tourne sur les ports :

- 8080 mappé sur 10081 (au niveau Host OS)
- 50000 mappé sur 50000 (au niveau Host OS)

Nous avons partagé le dossier qui contient les données les plus importants (`jenkins_home`) sur le Host OS, par contre pour des raisons que nous allons les sitées par la suite, nous avons partagé le `docker.sock` du Host OS vers le container Jenkins.

Pour assurer l'haute disponibilité de Jenkins, nous avons configuré le lancement automatique dans le cas du crash du service ou redémarrage du système d'exploitation.

➤ Sonar

```
sonar:
  image: webdizz/sonarqube:6.2
  ports:
    - 10082:9000
  depends_on:
    - mysql
  environment:
    - SONAR_JDBC_URL=jdbc:mysql://mysql:3306/sonarqube?useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=true
    - SONAR_DB_USERNAME=sonarqube
    - SONAR_DB_PASSWORD=sonarqube
  volumes:
    - ./sonar:/opt/sonar
  restart: unless-stopped
  container_name: sonar
```

Figure 19: Paramètres docker-compose pour sonar

Le container sonar est instancié d'une image docker-hub webdizz/sonarqube:6.2, il tourne sur le port 9000 mappé sur 10081(au niveau Host OS), il dépend d'un autre container : mysql, c'est-à-dire il se lance que si sa base mysql est déjà démarrée.

Pour se connecter sur mysql, sonar a besoin de trois variables d'environnement :

- SONAR_JDBC_URL
- SONAR_DB_USERNAME
- SONAR_DB_PASSWORD

Les données les plus importants pour ce container se trouvent sur /opt/sonar, nous avons protégé ce répertoire sur le système d'exploitation.

Sonar se relance automatiquement après chaque redémarrage du système hôte ou bien si son service crache.

➤ Mysql

```
mysql:
  image: mysql:5.6
  hostname: mysql
  expose:
    - 3306
  environment:
    - MYSQL_ROOT_PASSWORD=admin123
    - MYSQL_USER=sonarqube
    - MYSQL_PASSWORD=sonarqube
    - MYSQL_DATABASE=sonarqube
  volumes:
    - ./mysql:/var/lib/mysql
  restart: unless-stopped
  container_name: mysql
```

Figure 20: Paramètres docker-compose pour mysql

Le container mysql est instancié de l'image officielle mysql sur docker-hub, il tourne sur le port 3306, nous avons passé comme variables d'environnement :

- MYSQL_ROOT_PASSWORD
- MYSQL_USER
- MYSQL_PASSWORD
- MYSQL_DATABASE

Les trois derniers permettent la création de l'utilisateur sonarqube ainsi qu'une base de données pour le container sonar.

Le volume partagé avec le système hôte est /var/lib/mysql

Le container se relance automatiquement s'il y'a un redémarrage système ou un crache de son service.

➤ Nexus

```
nexus:
  image: sonatype/nexus
  ports:
    - "10083:8081"
  volumes:
    - ./nexus-data:/nexus-data
  restart: unless-stopped
  container_name: nexus
```

Figure 21: Paramètres docker-compose pour nexus

Le container nexus est instancié de l'image officielle sonatype/nexus sur docker-hub, il tourne sur le port 8081 mappé sur le port 10083 au niveau système hôte.

Les données sous /nexus-data sont partagées avec notre système d'exploitation.

Comme pour tous les autres composants, nous avons assuré l'haute disponibilité de nexus.

3.2.3. Manipulation technique via docker et docker-compose :

➤ Lancement des services :

Comme nous l'avons mentionné, docker-compose est l'outil responsable du lancement des services composant notre architecture.

Pour déployer l'architecture, une seule commande lancée par docker-compose doit mettre notre plateforme de l'intégration continue en place.

Ci-dessous le répertoire et son contenu où nous exécutons le script yaml (intégration.yml):

```
[root@ns3270619 persist]# ll
total 28
-rw-r--r--  1 root    root      1159 26 mai   13:31 intégration.yml
-rw-r--r--  1 root    root        566  7 mai   23:30 intégration.yml.backup
drwxr-xr-x 18 admin  admin    4096 30 mai   02:33 jenkins
drwxr-xr-x  5 polkitd ssh_keys 4096 30 mai   02:10 mysql
drwxr-xr-x 15 root    root     4096  8 mai   23:18 nexus-data
drwxr-xr-x 10 admin  admin    4096  7 mai   23:54 sonar
drwx----- 3 admin  admin    4096  8 mai   23:11 sonatype-work
[root@ns3270619 persist]# █
```

Figure 22: Répertoire du lancement de la plateforme

Nous prenons une copie du script comme source de backup.

Les dossiers : jenkins, mysql, nexus-data, sonar et sonatype-work sont les volumes partagés des containers vers l'hôte.

Ci-dessous la commande docker-compose pour lancer les services de la plateforme :

```
[root@ns3270619 persist]# docker-compose -f intégration.yml up -d
Creating network "persist_default" with the default driver
Creating jenkins ... done
Creating mysql ... done
Creating nexus ... done
Creating sonar ... done
[root@ns3270619 persist]# █
```

Figure 23: Lancement de la plateforme

En exécutant la commande ci-dessus, docker compose crée un nouveau réseau persist_default pour le cluster des conteneurs configurés dans le fichier intégration.yml, puis il lance l'ensemble des containers.

L'option -f indique le chemin vers le fichier yaml.

La commande up crée les services.

L'option -d : pour le lancement en background.

Après l'exécution de fichier ci-dessus, docker lit les noms des images configurées dans le fichier et les télécharge depuis le docker hub.

Une fois téléchargées, nous listons les images docker :

```
root@ns3270619:/home/abdou
[root@ns3270619 abdou]# docker image ls
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
mysql                5.6                5f5ccdc8aedc       5 weeks ago        256MB
jenkins              latest             7b210b6c238a       2 months ago        801MB
sonatype/nexus       latest             ee28e4c8635c       3 months ago        468MB
webdizz/sonarqube    6.2                65150ebf03b1       8 months ago        1.03GB
```

Figure 24: Liste des images docker téléchargées

Docker associe à chaque image un id, il affiche des informations comme la taille, la date de création et la version de l'image téléchargé.

➤ **Arrêt des services :**

Grâce aux volumes partagés, rien à perdre dans le cas où nous arrêterons les composants de notre solution, toutes les données importantes sont sauvegardées dans les dossiers de backup,

Une fois nous lançons de nouveau la plateforme, docker-compose charge le contenu des volumes dans les containers.

Ci-dessous la commande docker-compose pour arrêter les services :

```
[root@ns3270619 persist]# docker-compose -f intégration.yml down
Stopping sonar      ... done
Stopping mysql     ... done
Stopping jenkins   ... done
Stopping nexus     ... done
Removing sonar     ... done
Removing mysql     ... done
Removing jenkins   ... done
Removing nexus     ... done
Removing network persist_default
[root@ns3270619 persist]#
```

Figure 25: Suppression de la plateforme

La commande dessus s'exécute en trois étapes :

- Arrêt des containers
- Suppression des containers
- Suppression du réseau créé lors du lancement des services

➤ **Liste des composants de la plateforme :**

Docker-compose nous offre la possibilité de lister nos services au cours d'exécution à travers la commande suivante :

```
[root@ns3270619 persist]# docker-compose -f intégration.yml ps
Name                Command                State                Ports
-----
jenkins             /bin/tini -- /usr/local/bi ... Up                0.0.0.0:50000->50000/tcp, 0.0.0.0:10081->8080/tcp
mysql               docker-entrypoint.sh mysqld Up                3306/tcp
nexus               /bin/sh -c ${JAVA_HOME}/bi ... Up                0.0.0.0:10083->8081/tcp
sonar               /bin/sh -c /opt/sonar/bin/ ... Up                0.0.0.0:10082->9000/tcp
[root@ns3270619 persist]#
```

Figure 26: Liste des composants par docker-compose

Nous pouvons aussi lister les containers qui tournent par docker engine comme suit :

```
[root@ns3270619 persist]# docker ps
CONTAINER ID        IMAGE                COMMAND                CREATED                STATUS                PORTS                NAMES
9bf6189abd8c       webdizz/sonarqube:6.2 "/bin/sh -c '/opt/so..." 2 hours ago           Up 2 hours           0.0.0.0:10082->9000/tcp        sonar
99f5f4edc6cc       jenkins             "/bin/tini -- /usr/l..." 2 hours ago           Up 2 hours           0.0.0.0:50000->50000/tcp, 0.0.0.0:10081->8080/tcp        jenkins
18a35b19de42       sonatype/nexus     "/bin/sh -c '${JAVA_..." 2 hours ago           Up 2 hours           0.0.0.0:10083->8081/tcp        nexus
db62912037ae       mysql:5.6           "docker-entrypoint.s..." 2 hours ago           Up 2 hours           3306/tcp                mysql
[root@ns3270619 persist]#
```

Figure 27: Liste des composants par docker engine

La structure du produit YessClever est générée par maven, pour tourner son intégration continue, il faut installer maven avec jenkins.

Ci-dessous les deux lignes commandes d'installation de maven dans le conteneur jenkins :

```
root@ns3270619:/home/abdou
[root@ns3270619 abdou]# docker exec -it -u root jenkins apt-get update
[root@ns3270619 abdou]# docker exec -it -u root jenkins apt-get install maven
[root@ns3270619 abdou]#
```

Figure 28: Installation du maven dans jenkins

La première commande sert à mettre à jour le dépôt APT, alors que la deuxième installe le package maven.

➤ Configuration de jenkins :

Après l'exécution du script yaml, nous configurons jenkins pour créer le premier utilisateur administrateur.

Avec la commande « **docker logs -f jenkins** », nous détectons le mot de passe d'initialisation de jenkins comme suit :

```
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

c4bd13de71834cbe9d14f57bf8696974

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*****
*****
*****

May 02, 2018 10:12:18 PM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
May 02, 2018 10:12:18 PM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
--> setting agent port for jnlp
--> setting agent port for jnlp... done
```

Figure 29: Copie du mot de passe de verrouillage de jenkins

Nous copions par la suite ce mot de passe, pour avoir l'accès en tant qu'administrateur sur l'interface jenkins.

L'étape suivante est la création du premier utilisateur jenkins avec le rôle l'administrateur.

Démarrage

Créer le 1er utilisateur Administrateur

Nom d'utilisateur:

Mot de passe:

Confirmation du mot de passe:

Nom complet:

Adresse courriel:

Jenkins 2.60.3 Continuer en tant qu'Administrateur [Sauver et Terminer](#)

Figure 30: Création du premier utilisateur jenkins

Après la création de l'utilisateur administrateur, jenkins sera prêt pour passer à une autre phase de configuration plus avancée.

3.2.4. Configuration avancé de jenkins :

Le serveur jenkins va gérer toute la chaîne de l'intégration continue, ce qui nécessite sa configuration avec les autres outils de l'architecture :

➤ Jenkins avec sonar :

- Génération du token au niveau sonar :

Le token généré au niveau sonar sera configuré par la suite dans jenkins.

- Ajout du plugin sonargube scanner sous jenkins :

Ce plugin offre la possibilité d'ajouter des serveurs sonar pour l'analyse du code source de n'importe quel projet.

- Configuration au niveau jenkins :

Figure 31: Configuration jenkins-sonarqube

Dans l'interface ci-dessous, nous configurons jenkins pour qu'il soit capable de faire appel au sonar afin d'analyser notre projet YessClever.

➤ Jenkins avec Nexus

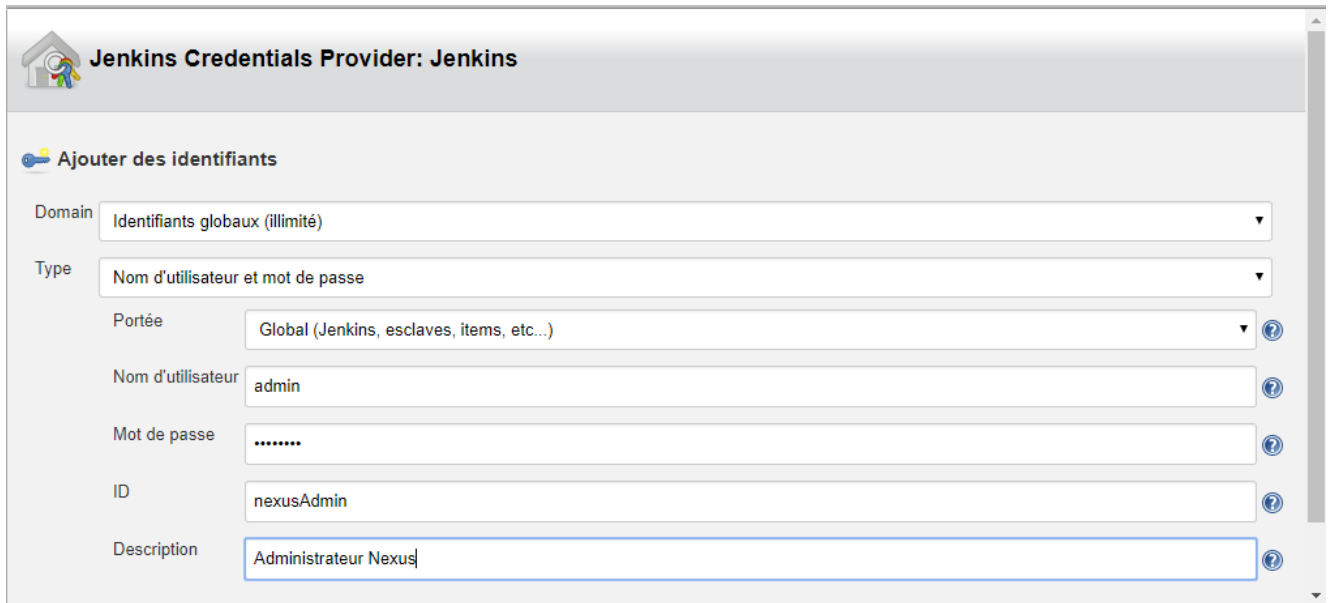
Navigation : Administrer Jenkins → Gestion des plugins → Avancé

Soumettre un plugin :

Figure 32: Interface jenkins pour soumettre un plugin

Nous cliquons sur « choisir un fichier » pour ajouter le plugin Nexus depuis notre dossier de téléchargement.

Nous allons ajouter l'utilisateur admin du serveur nexus sous jenkins pour qu'ils puissent communiquer.

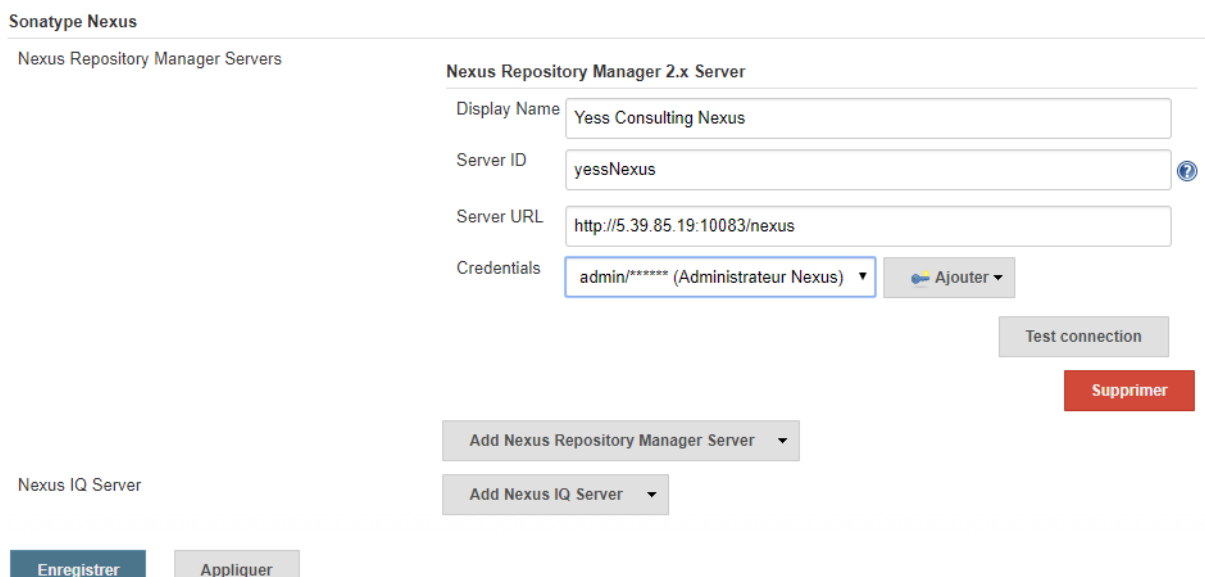


The screenshot shows the 'Jenkins Credentials Provider: Jenkins' interface. Under the heading 'Ajouter des identifiants', there are several form fields: 'Domain' set to 'Identifiants globaux (illimité)', 'Type' set to 'Nom d'utilisateur et mot de passe', 'Portée' set to 'Global (Jenkins, esclaves, items, etc...)', 'Nom d'utilisateur' set to 'admin', 'Mot de passe' masked with dots, 'ID' set to 'nexusAdmin', and 'Description' set to 'Administrateur Nexus'.

Figure 33: Création d'un utilisateur nexus sous jenkins

L'étape suivante est :

Navigation : Administrer Jenkins → Configurer le système → Sonatype Nexus



The screenshot shows the 'Sonatype Nexus' configuration page. Under 'Nexus Repository Manager Servers', there is a section for 'Nexus Repository Manager 2.x Server'. The fields are: 'Display Name' (Yess Consulting Nexus), 'Server ID' (yessNexus), 'Server URL' (http://5.39.85.19:10083/nexus), and 'Credentials' (admin/***** (Administrateur Nexus)). There are buttons for 'Ajouter', 'Test connection', and 'Supprimer'. Below this, there are buttons for 'Add Nexus Repository Manager Server' and 'Add Nexus IQ Server'. At the bottom, there are 'Enregistrer' and 'Appliquer' buttons.

Figure 34: Configuration jenkins-nexus

Ci-dessous, nous configurons les trois champs principaux :

- L'ID du serveur : yessNexus

Ça sera configuré dans notre script yaml pour le déploiement des artefacts sous Nexus.

- L'URL du serveur : le lien du serveur Nexus
- Credentials : l'utilisateur admin créé dessus.

3.3. Réalisation fonctionnelle

Après avoir configuré la partie technique, nous entamons la partie fonctionnelle.

Cette partie consiste à configurer l'orchestrateur jenkins pour communiquer avec les autres composants de notre architecture afin d'assurer le bon fonctionnement de l'intégration continue.

Ci-dessous l'interface principale jenkins dès son installation :

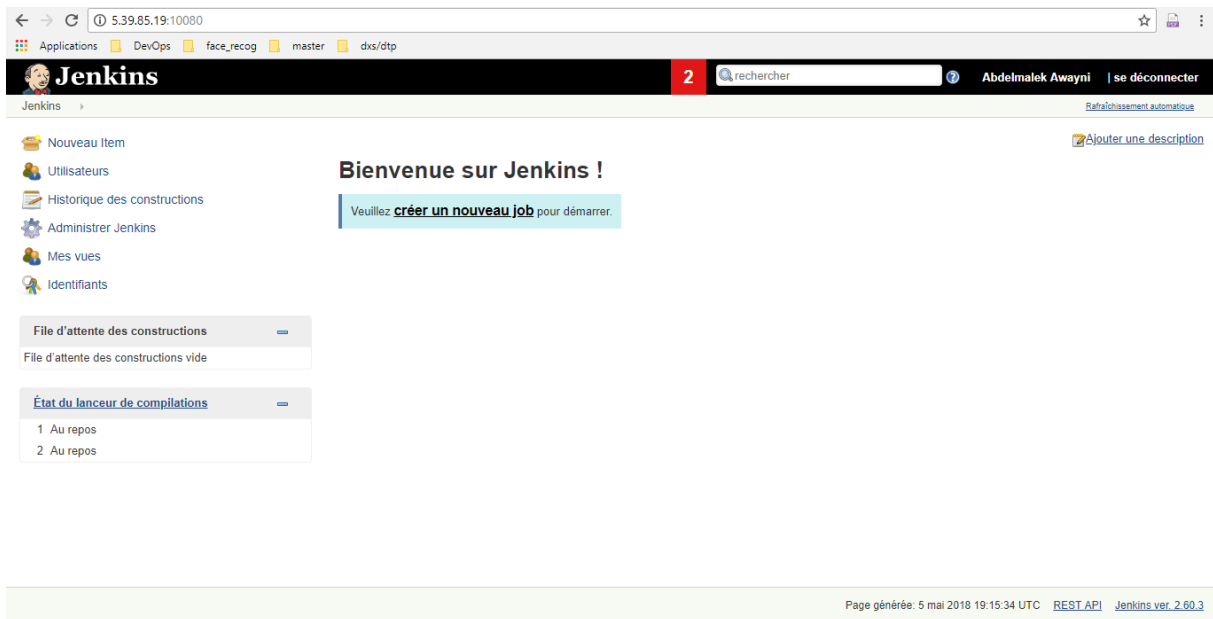


Figure 35: Interface principale de jenkins

Création d'un nouveau job jenkins de type pipeline pour tourner le projet YessClever :

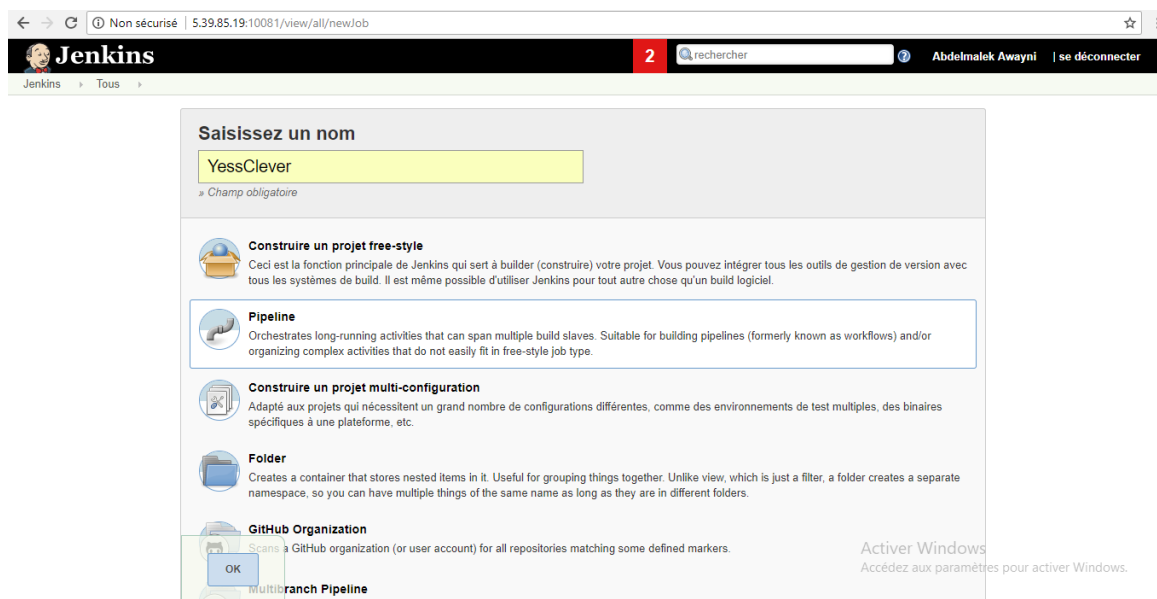


Figure 36: Création d'un nouveau projet jenkins

Nous avons saisi le nom du job : YessClever

Nous cochons par la suite le type du job : Pipeline

En fin nous cliquons sur OK

Configuration du job

Comme nous avons mentionné dans la partie état de l'art, nous utilisons le langage groovy pour configurer notre job YessClever.

Il s'agit d'un script qui se compose de plusieurs « stage »

Ci-dessous l'interface de configuration du job YessClever :

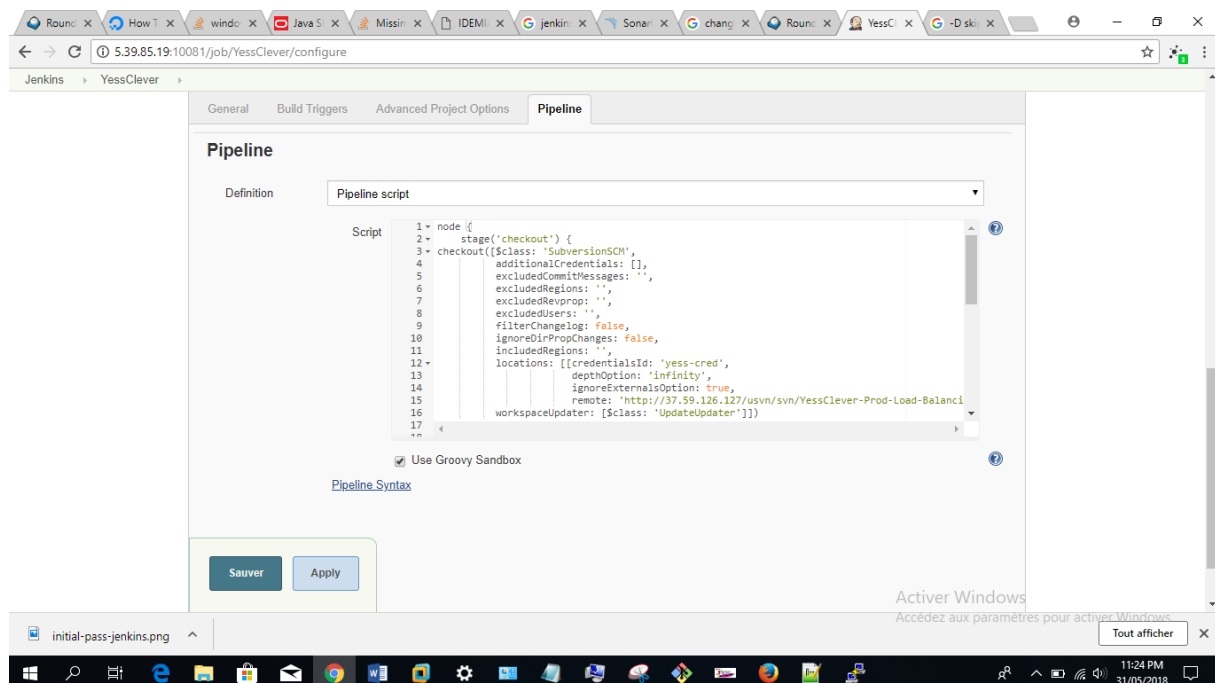


Figure 37: Script groovy pour le pipeline jenkins

Ce qui nous concerne principalement est la partie Pipeline :

La procédure de l'intégration continue se fait sur plusieurs étapes(ou bien stage) :

- Stage « **checkout** » : ceci communique avec le gestionnaire de source SVN pour télécharger le code source.

Pour se faire, nous devons ajouter l'accès jenkins sur SVN, Yess Consulting nous crée spécialement un compte SVN.

Ci-dessous nous montrons comment ça se passe :

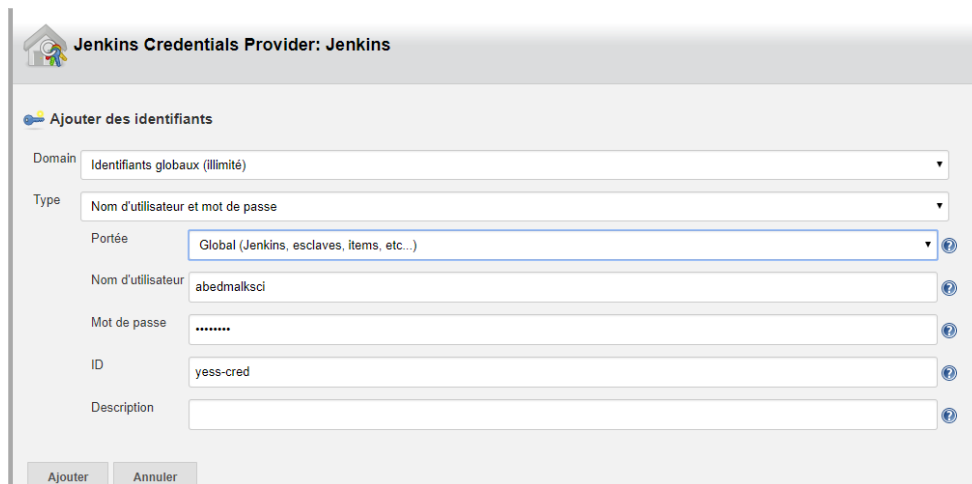


Figure 38: Création d'un utilisateur SVN pour le téléchargement du projet

Après avoir configuré l'accès à SVN, nous lançons le premier build pour tester le bon fonctionnement du stage « **checkout** » :

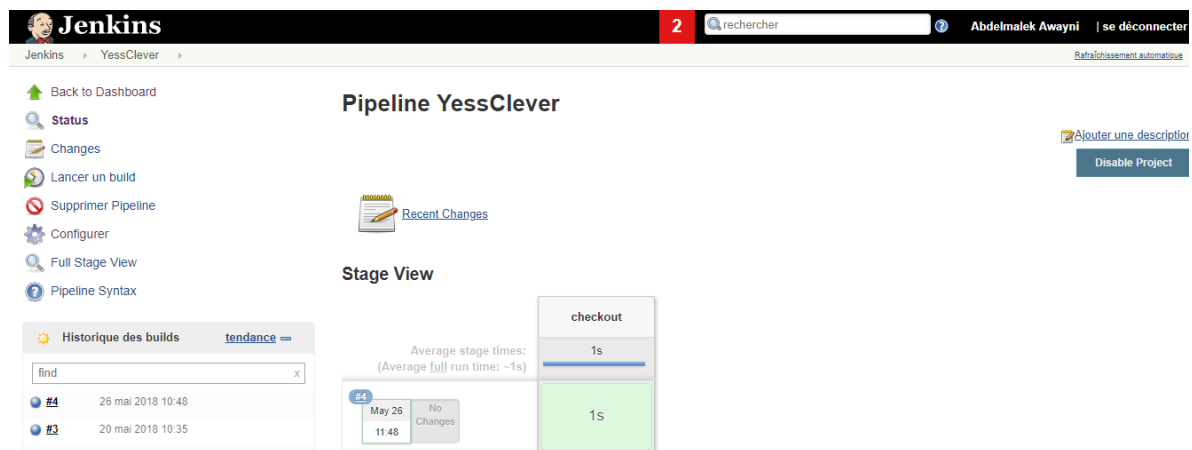


Figure 39: Téléchargement du projet YessClever depuis SVN

La couleur verte affichée à la fin de chaque lancement du build indique la réussite du stage en testé, en effet le téléchargement du projet YessClever est terminé avec succès.

Dans la sortie de console, le pipeline jenkins nous affiche le log qui concerne le téléchargement du projet depuis SVN :

Sortie de la console

```

Started by user Abdelmalek Awayni
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/YessClever
[Pipeline] {
[Pipeline] stage
[Pipeline] { (checkout)
[Pipeline] checkout
Updating http://37.59.126.127/usvn/svn/YessClever-Prod-Load-Balancing/trunk/YessClever at revision '2018-06-10T11:42:54.156 +0000'
Using sole credentials abedmalksci/***** in realm '<http://37.59.126.127:80> "USVN"'
At revision 13

No changes for http://37.59.126.127/usvn/svn/YessClever-Prod-Load-Balancing/trunk/YessClever since the previous build

```

Figure 40: Log correspond au stage du téléchargement

- Stage « **check java** » : la vérification de l'installation java, si la réponse est négative, le build est d'état « FAILURE », puisque jenkins va exécuter des commandes maven, et ce dernier a besoin de java.

Stage View

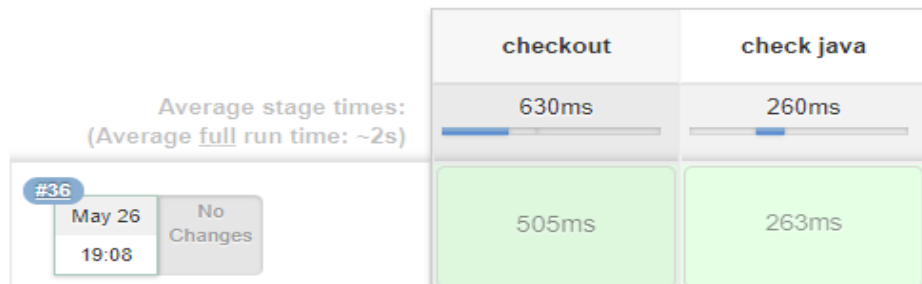


Figure 41: Vérification de la bonne installation java

Jenkins confirme à travers ce stage, la bonne installation de java, dans la console des logs il nous affiche la version de java :

```

[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (check java)
[Pipeline] sh
[YessClever] Running shell script
+ java -version
openjdk version "1.8.0_162"
OpenJDK Runtime Environment (build 1.8.0_162-8u162-b12-1~deb9u1-b12)
OpenJDK 64-Bit Server VM (build 25.162-b12, mixed mode)

```

Figure 42: Affichage de la version java

- Stage « **clean** » : ceci supprime l'historique du build précédent, il met l'espace de travail en état initial.

L'action clean est lancée par la commande maven : **mvn clean**

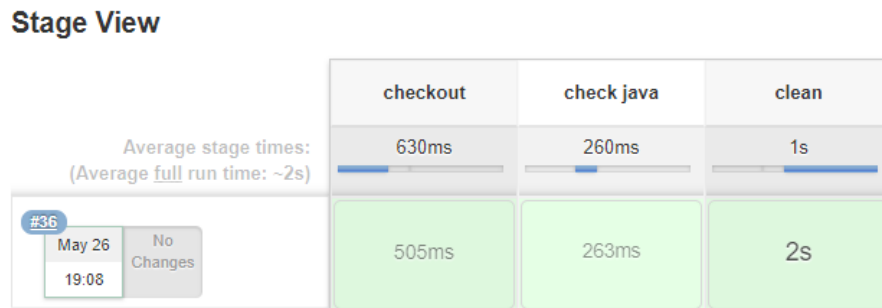


Figure 43: Stage de la suppression de l'espace du travail

Le nettoyage de l'espace de travail est réalisé avec succès, après cette étape, jenkins commence à compiler le code source de notre projet YessClever.

Ci-dessous l'affichage sur la console des logs spécifique à ce stage :

```
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (clean)
[Pipeline] tool
[Pipeline] withEnv
[Pipeline] {
[Pipeline] sh
[YessClever] Running shell script
+ cd YessClever
+ mvn clean
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for YessClever:YessClever:war:0.0.1
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 555, column 12
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-war-plugin is missing. @ line 548, column 12
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] -----
[INFO] Building YessClever 0.0.1
[INFO] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ YessClever ---
[INFO] Deleting /var/jenkins_home/workspace/YessClever/YessClever/target
[INFO] Deleting /var/jenkins_home/workspace/YessClever/YessClever/WebContent/WEB-INF/classes
[INFO] -----
[INFO] BUILD SUCCESS
-----
```

Figure 44: Sortie de la console pour le stage « clean »

- Stage « **test** » : c'est le stage le plus important car à travers ce dernier, jenkins exécute la compilation du code source. En cas d'erreur il met build en état « **FAILURE** » et notifie les développeurs.

L'action test est lancée par la commande : **mvn test**

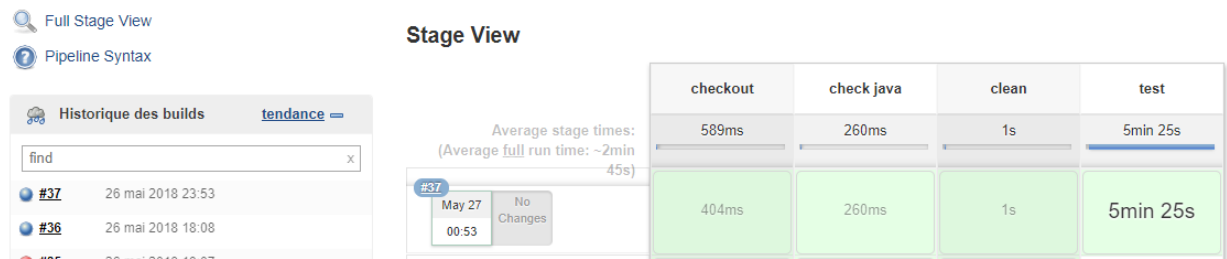


Figure 45: Compilation du code source

Ci-dessus, le pipeline jenkins montre la fin de la compilation du code source avec succès.

- Stage « **sonar scan** » : cette étape est très utile pour le développeur, en effet sonar analyse le code source du projet et génère tout un rapport contenant des détails concernant la qualité de code.

L'action sonar test est lancée par la commande maven : **mvn sonar:sonar**

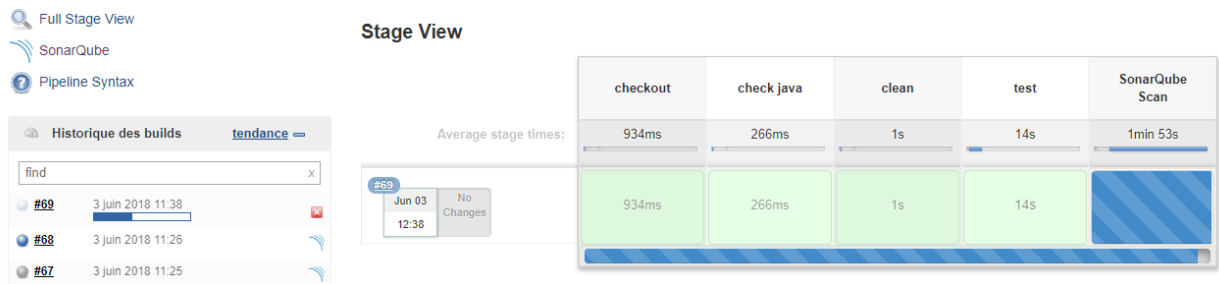
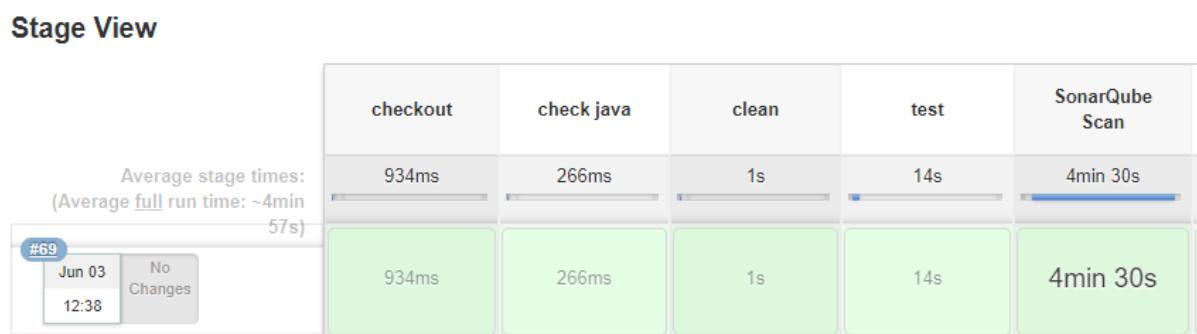


Figure 46: Scan du projet avec sonarqube

A la fin de ce stage, le pipeline jenkins affiche le résultat des tests réalisés par sonarqube. Dans notre cas, le projet YessClever a réussi le test de la couverture de code.



SonarQube Quality Gate

YessClever **OK**
 server-side processing: **Success**

Figure 47: Déroulement du stage SonarQube Scan

Ci-dessous l'interface principale du rapport généré par sonar :

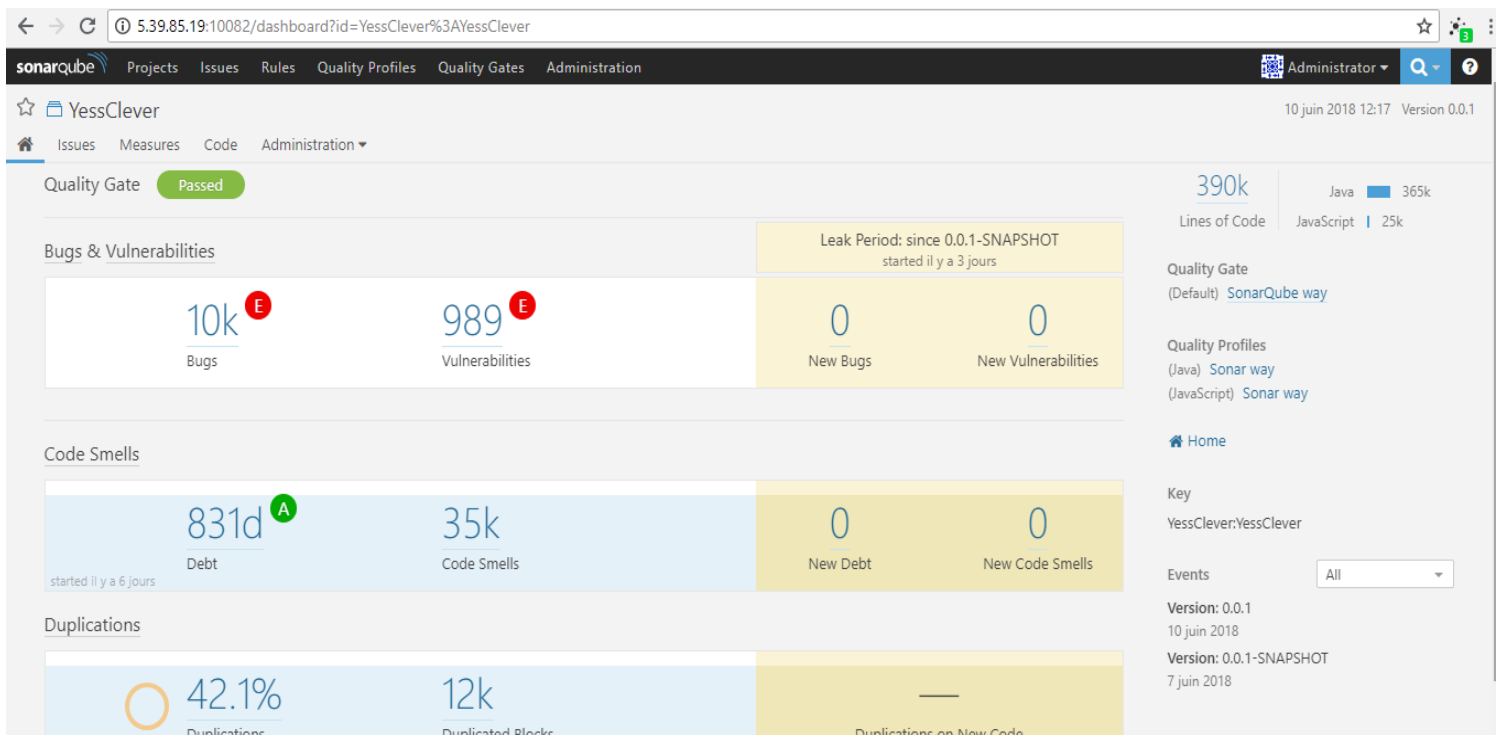


Figure 48: Interface sonarqube pour le projet YessClever

- Stage « **package** » : si les stages précédents sont exécutés avec succès, jenkins lance une commande maven pour créer la livrable du projet c'est-à-dire le .war qui va être déployé dans le serveur applicatif (Tomcat dans notre cas)

L'action package est lancée par la commande maven : **mvn package**

Stage View

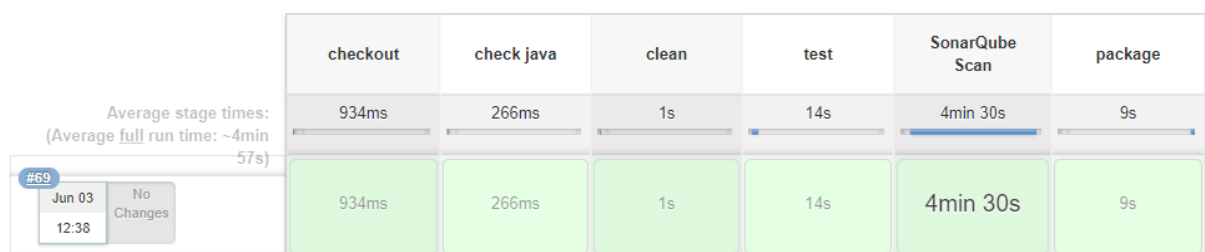


Figure 49: Génération du livrable

La génération du livrable est faite avec succès.

- Stage « **Publich** » : si les stages précédents ont été tous réussis, le pipeline jenkins envoi la livrable générée dans le stage précédent vers le gestionnaire de dépôt Nexus.

Stage View



Figure 50: Envoi du livrable vers nexus

Nous avons configuré notre pipeline pour lire les informations nécessaires depuis le fichier pom.xml avant de publier l’artefact dans Nexus.

Les principales informations sont :

- artifactId : YessClever
- groupId : YessClever
- version : 0.0.1

Ci-dessous l’artefact publié par jenkins dans Nexus : YessClever-0.0.1.war

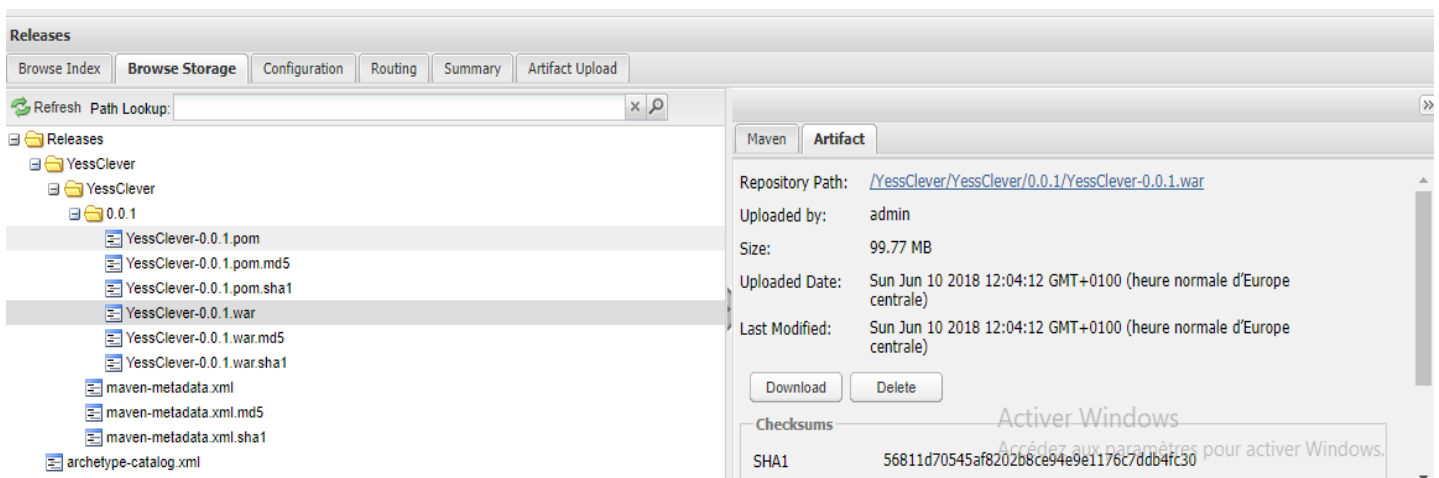


Figure 51: Livrable sous nexus

Il est possible de télécharger la livrable manuellement, comme il est possible de la déployer dans serveur applicatif (tomcat, jboss, ...) d’une manière automatisée.

3.4. Conclusion :

Au cours de ce dernier chapitre, nous avons implémenté notre solution qui consiste en la mise en place d'une plateforme d'intégration continue, et qui a été déroulé sur deux étapes : une étape technique pour la configuration de l'environnement, et une étape fonctionnelle pour la configuration des composants.

Conclusion générale

Ce travail s'inscrit dans le cadre de la réalisation du projet de fin d'études, il est réalisé au sein de la société Yess Consulting qui consiste à la mise en place d'une chaîne d'intégration continue basée sur docker et docker-compose. Nous avons décomposé notre travail sur deux modules, un module concernant l'infrastructure sur le serveur et un module d'intégration continue.

Le choix de la technologie de virtualisation Docker met le déploiement de notre solution très rapide grâce à son outil docker-compose. En effet, avec une ligne de commande, tous les composants de l'architecture seront déployés dans quelques secondes.

La rapidité de déploiement de notre architecture présente une grande importance dans notre projet, en effet, nous pouvons tester n'importe quel logiciel en mode de développement dans un intervalle de temps très court.

Le déploiement de la plateforme, ainsi que les tests que nous avons fait pour le produit YessClever sont tous réalisés avec succès.

Ce stage nous a beaucoup apporté, tant sur le plan personnel que professionnel. Il nous a permis de mieux comprendre et d'appréhender l'organisation d'une entreprise, ses contraintes et ses missions. Ceci présente pour nous une opportunité pour mieux comprendre le concept du DevOps.

Liste des Références

- [1] Récupéré sur : <http://www.yess-consulting.com/>
- [2] Récupéré sur : <https://artkeos.com/qu-est-ce-que-integration-continue/>
- [3] Récupéré sur : <http://www.isi.rnu.tn/BIBI/ing/P%20ING%20513>
- [4] Récupéré sur : <http://www.arcitek.fr/?fond=virtualisation>
- [5] Récupéré sur : <http://www.it-connect.fr/les-types-dhyperviseurs/>
- [6] Récupéré sur : <https://jack-vanlightly.com/blog/2017/2/19/sonarqube-hell-yes-sonarlint-no-thanks>
- [7] Récupéré sur : <http://blog.nicolargo.com/2014/06/virtualisation-legere-docker.html>

Annexe

Installation docker :

```
sudo yum check-update
```

```
curl -fsSL https://get.docker.com/ | sh
```

```
sudo systemctl start docker
```

```
sudo systemctl status docker
```

```
sudo systemctl enable docker
```

Vérification de l'installation :

```
abdou@ns3270619:~  
[abdou@ns3270619 ~]$ docker -v  
Docker version 18.03.1-ce, build 9ee9f40  
[abdou@ns3270619 ~]$
```

Par défaut, les commandes docker s'exécutent seulement avec l'utilisateur root

Un utilisateur normal n'a pas les permissions nécessaires pour exécuter les commandes docker qu'après l'ajouter dans le groupe docker à travers la commande suivante :

```
sudo usermod -aG docker $(whoami)
```

Installation docker-compose :

```
sudo yum install epel-release
```

```
sudo yum install -y python-pip
```

```
sudo pip install docker-compose
```

Vérification de l'installation :

```
abdou@ns3270619:~  
[abdou@ns3270619 ~]$ docker-compose -v  
docker-compose version 1.21.0, build 5920eb0  
[abdou@ns3270619 ~]$
```