Table des matières

Liste des tableaux

Liste des figures

Introduction générale

L'avènement de l'entreprise numérique implique la révision profonde des modes de création d'application. Il n'est plus possible de patienter 6 mois pour les livrables de développement. Avec ces exigences de temps au marché et l'évolution des projets dans des configurations logicielles et d'infrastructure de plus en plus complexes, générant des risques opérationnels et de planification, les besoins d'industrialiser les tests et de fluidifier les déploiements en production se sont progressivement affirmés. En rapprochant les équipes de développement, de test et d'exploitation, le DevOps répond précisément à ce défi du digital. Les entreprises en ont désormais bien conscience.

Parmi les pratiques les plus répandues de DevOps, on trouve la livraison continue et la gestion de configuration. La livraison continue est une stratégie logicielle qui permet aux organisations d'offrir de nouvelles fonctionnalités aux utilisateurs rapidement et efficacement. L'idée de base de livraison continue est de créer un processus reproductible et fiable d'amélioration progressive pour amener le logiciel du concept au client. L'objectif de la livraison continue est de permettre un flux constant de changements vers la production via une ligne de production automatisée de logiciels. Le pipeline de livraison continue est ce qui rend tout cela possible.

C'est dans ce cadre s'inscrit le présent projet, qui vise à améliorer un pipeline de livraison continue en intégrant Ansible, un outil de gestion de configuration et d'automatisation. Ce projet s'est déroulé au sein de l'entreprise SQLI, qui vise à régler plusieurs problématiques.

Le présent rapport décrit l'essentiel du travail réalisé lors de ce projet, il est organisé en quatre chapitres :

Dans le premier chapitre, nous présentons l'entreprise d'accueil SQLI, après nous explorons la démarche DevOps chez SQLI. Puis, nous poursuivons avec une étude de l'existant dont en déduisent les problèmes. Après, nous exposons notre solution et la démarche suivie pour la gestion de projet.

Dans le deuxième chapitre, nous présentons une étude comparative des outils DevOps les plus répandus sur le marché, ainsi que l'outil choisi dans chaque catégorie.

Dans l'avant-dernier chapitre, analyse et conception, nous exposerons les besoins fonctionnels, les acteurs, les cas d'utilisations et l'architecture physique des environnements constituants du pipeline.

Le dernier chapitre est consacré pour présenter les tâches réalisées pour implémenter notre projet.

Chapitre 1 : Cadre générale du projet

Introduction

Ce chapitre a pour objectif de situer notre travail par rapport à son cadre général. Nous commençons tout d'abord par la présentation de l'entreprise d'accueil SQLI dans lequel s'est déroulé notre stage, puis nous passons par une étude de l'existant qui permettra de faire une évaluation et critique de ce dernier. Enfin nous présenterons nos solutions, ainsi que la démarche de travail suivie.

1. Entreprise d'accueil

Nous donnons dans la première partie de ce chapitre quelques informations sur le groupe SQLI et nous décrivons ses secteurs d'activités et ses différents pôles. Ainsi qu'une vue organisationnelle de SQLI Maroc et une description du projet AMC.

1.1. Le groupe SQLI

Créé en 1990, SQLI est le partenaire de référence des entreprises et des marques dans la transformation digitale de leur parcours client et de tous les services internes impactés par cette évolution.

Son positionnement unique au confluent du marketing et de la technologie lui permet de répondre de façon globale aux enjeux de développement des ventes et de notoriété (marketing digital & social, expérience client, commerce connecté, data intelligence...) ainsi qu'aux enjeux de productivité et d'efficacité interne (digitalisation des opérations, entreprise collaborative, mobilité et objets connectés, CRM...).

Ses 2400 collaborateurs sont répartis en France (Paris, Lyon, Toulouse, Bordeaux, Rouen, Nantes et Lille), en Suisse (Lausanne et Genève), au Luxembourg, en Belgique (Bruxelles et Gand), au Royaume-Uni (Londres), au Maroc (Rabat et Oujda) et en Afrique du Sud (Cape Town). La figure 1 représente l'empreinte européenne du groupe, ainsi une capacité de déploiement global.

Figure. 1 : Répartition des agences SQLI dans le monde

Le Groupe SQLI a réalisé en 2015 un chiffre d'affaires de 179 M€. Depuis le 21 juillet 2000, la société SQLI est cotée sur Euronext Paris (SQI).

1.2. Clients SQLI

Tout en développant son activité, SQLI veille à maintenir une grande diversification de sa clientèle et des secteurs d'activités auxquels elle s'adresse, de façon à contenir le risque de concentration sur un nombre restreint de clients.

SQLI compte plus de 1200 clients, grands comptes et PME (petite et moyenne entreprise), issus de tous les secteurs d'activité (aéronautique, finance, assurance, industrielle, pharmaceutique), avec environ 90 % de clients fidèles.

Figure. 2 : Quelques clients du groupe SQLI

1.3. Partenaires SQLI

Le groupe a tissé des relations privilégiées avec les acteurs majeurs de la technologie et du logiciel, mais aussi avec de plus petits acteurs, chaque fois que leurs solutions présentent une vraie valeur ajoutée pour les clients. La figure 3 présente les principaux partenaires de SQLI.

Figure. 3 : Quelques partenaires de SQLI

1.4. Organisation SQLI Maroc

Créée il y a 13 ans, SQLI Maroc accompagne les grandes entreprises et les collectivités dans leur transformation digitale, depuis le conseil métier jusqu'à la mise en œuvre et la maintenance de dispositifs innovants.

Très ancrée localement, l'agence s'est également développée à l'international et compte désormais 60 % de clients offshore.

Ses équipes s'appuient sur une expertise de pointe des technologies pour :

- améliorer les ventes de ses clients : marketing digital & social, e-commerce, data marketing, UX et nouveaux usages ;
- et renforcer leur performance interne : conception de socles technologiques, Internet, Intranet, conseil et solutions Big Data, solutions mobiles.

SQLI Maroc rassemble plus de 600 collaborateurs, répartis à Casablanca, Rabat et Oujda.

Innovative service center (ISC) Maroc est organisée de manière à gagner en synergie entre ses deux sites marocains et de donner une lecture plus simple de son organisation. Les ISC de Oujda et Rabat constituent une seule entité « ISC Maroc ».

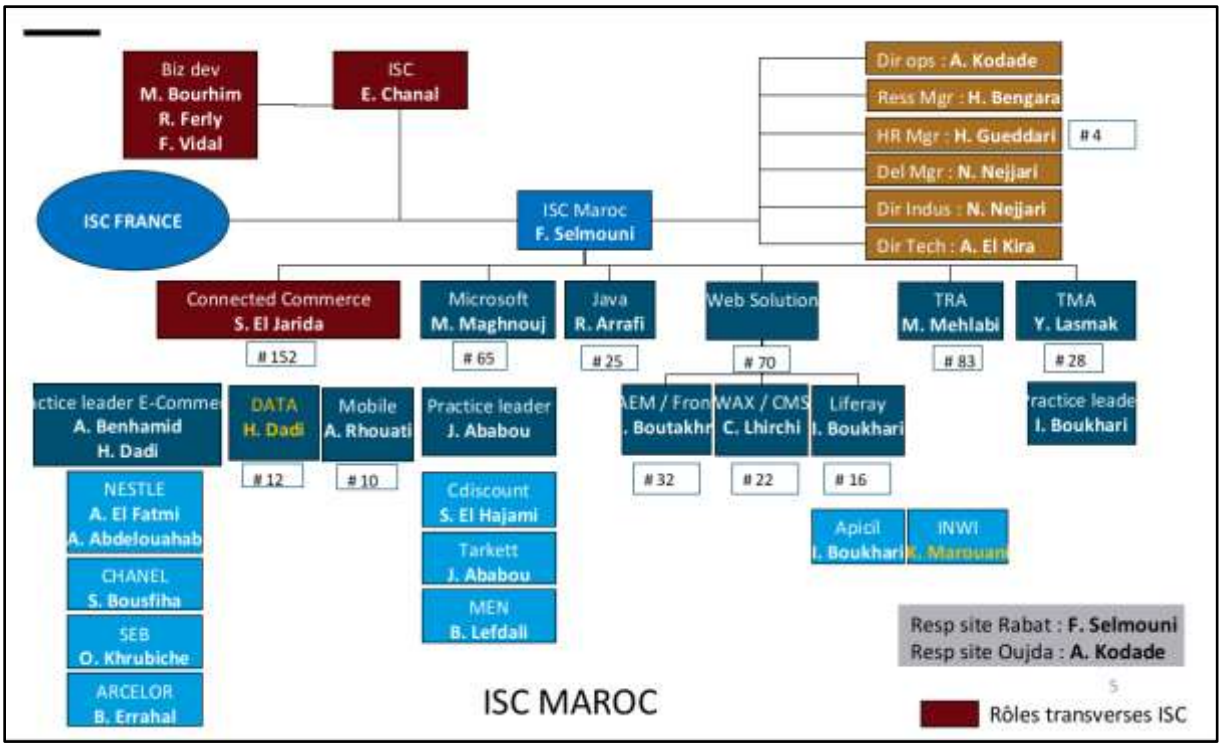


Figure. 4 : Organigramme ISC Maroc

Les deux sites sont sous la responsabilité du Directeur Delivery (livraison) qui a un focus des livraisons global afin de s'assurer que ISC Maroc produit un delivery de haute qualité pour les clients avec des moyens de production adéquats.

ISC Maroc est composée de 5 centres de compétences (Skills Centers) :

- E-Commerce/Java
- Microsoft (.NET & SharePoint)
- CMS/Front/Mobile
- Business Solutions (CRM & BI)
- TMA (Tests et Maintenance Applicative)

En plus de ces Skills Centers, ISC Maroc dispose aussi de centres de services dédiés à des clients (Nespresso, Airbus, etc.) sous la responsabilité de Services Managers.

1.5. Projet AMC

Notre travail de fin d'études sera appliqué sur le projet AMC du client ArcelorMittal dans une première étape. Alors que l'objectif final d'étaler notre travail sur l'ensemble des projets chez SQLI Maroc.

ArcelorMittal est le numéro un mondial de l'exploitation sidérurgique et minière, avec 199.000 salariés, une présence dans 60 pays et une empreinte industrielle dans 19 pays. ArcelorMittal est le principal fournisseur d'acier de qualité des grands marchés sidérurgiques mondiaux, y compris l'automobile, la construction, l'électroménager et l'emballage, soutenus par un département de recherche et développement d'envergure mondiale et d'excellents réseaux de distribution.

Le projet AMC est transformation numérique des activités d'ArcelorMittal de vente par l'appropriation d'une plateforme e-commerce B2B en se basant sur la solution SAP Hybris. Ce projet a pour but d'élargir la clientèle de base du client.

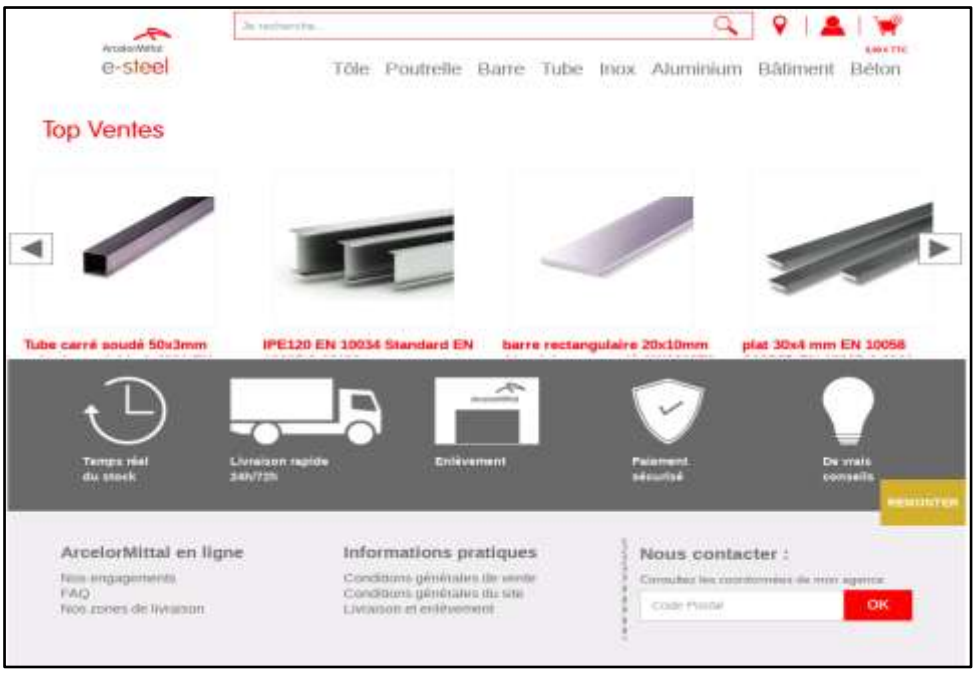


Figure. 5 : Page d'Accueil e-steel ArcelorMittal

2. DevOps chez SQLI

Notre projet de fin d'études s'intègre dans la démarche DevOps chez SQLI. Pour cela, dans cette partie nous allons expliquer l'offre DevOps, pourquoi et comment démarrer une démarche DevOps.

2.1. Pourquoi DevOps ?

DevOps est un ensemble de bonnes pratiques pour l'industrialisation du système d'information, plus une stratégie pour réduire le Time To Market.

Les méthodes agiles sont des formes efficaces de développement et gestion de projet qui offrent plusieurs avantages :

- Une haute qualité du produit ou service
- La prise en compte des besoins Métier, utilisateurs de manière continue
- Des livraisons régulières des nouvelles fonctionnalités (développements réguliers).

Mais, il reste encore des points d'amélioration afin d'étendre les principes agiles jusqu'aux équipes d'exploitation [1].

- Les mises en production (donc mises à disposition du produit aux utilisateurs) dépendent de plans de déploiement, de releases planifiées tous les mois, 3 mois, 6 mois, etc.
- Une fois en production, le produit n'est que peu étudié afin de s'assurer qu'il correspond réellement aux attentes des utilisateurs finaux. On note une faible réactivité face aux retours.

Pour ces raisons et d'autres, nous sommes confronté à un besoin d'étendre les principes agiles sur toute la chaîne de création d'un produit ou service informatique. Ce qui permettra de :

- Réduire le délai de mise en production, de mise en ligne, de mise sur le marché du produit
- Pouvoir livrer et démontrer à tout moment (livraison continue) l'état des produits [cf. Figure 6]
- Rapprocher les métiers du développement et des opérations (exploitation) pour une meilleure performance
- Être plus efficient tout au long du cycle de vie du produit
- Gérer efficacement et de façon plus simple les environnements via l'automatisation.

En vertu de ce qui précède, une comparaison des méthodes agile (sans DevOps) et méthode agile avec DevOps se résume dans la possibilité de livrer n'importe quel moment tant que les tests sont valides, au lieu de la fin de chaque sprint [cf. Figure 7].

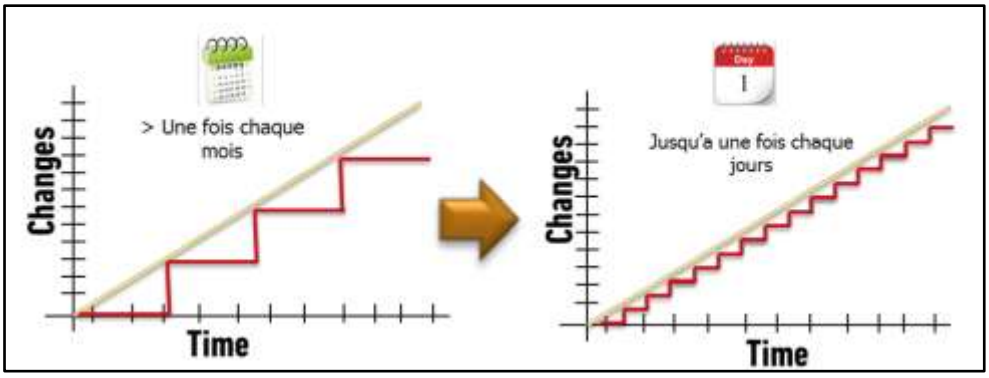


Figure. 6 : La possibilité de livrer plus fréquemment avec DevOps

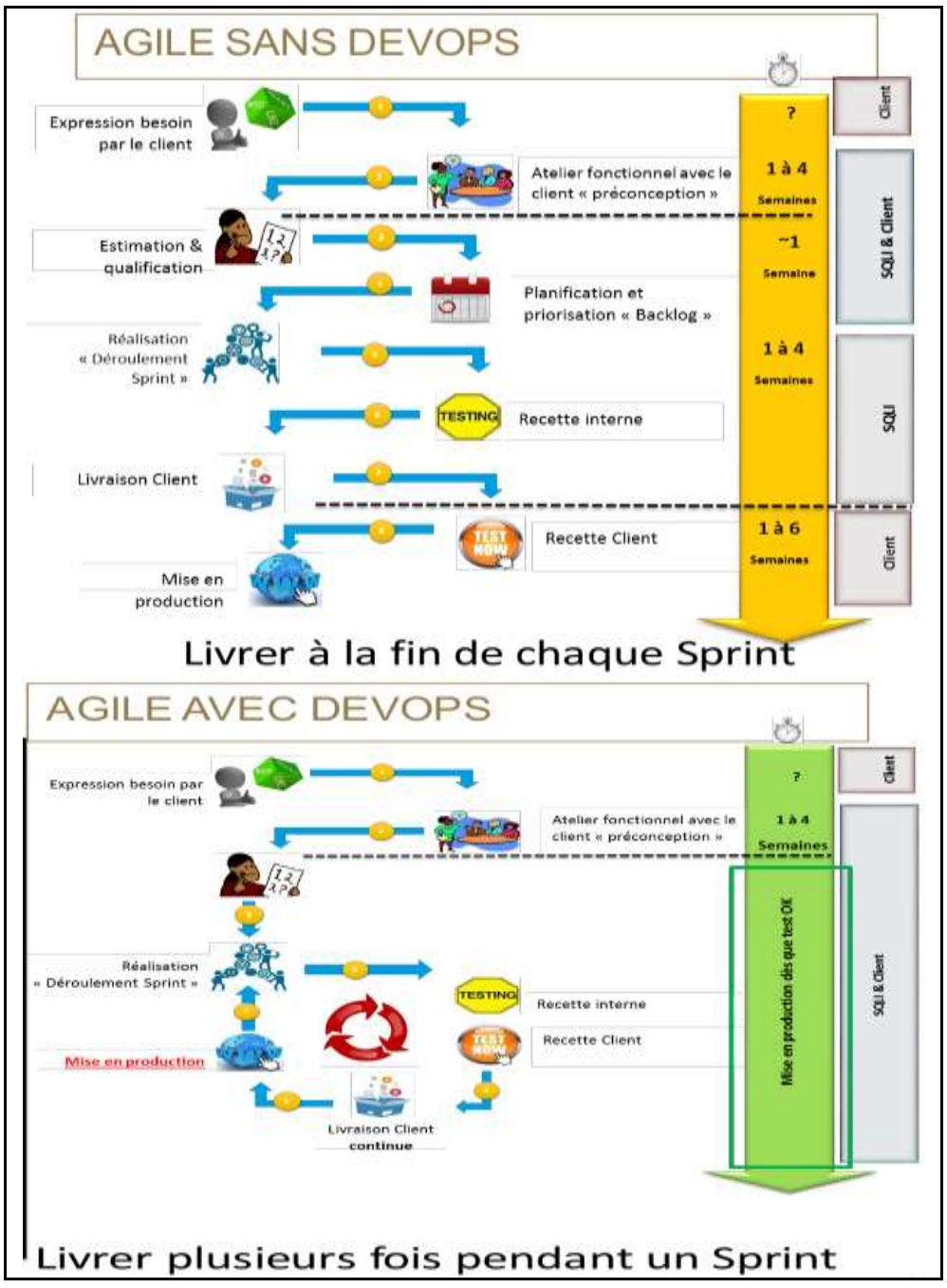


Figure. 7 : Différence entre une méthode agile sans et avec DevOps

2.2. Comment démarrer une transformation DevOps ?

Après que nous avons vu les qualités d'avoir une combinaison d'agilité et DevOps. Dans ce qui suit, nous montrons la démarche suivie par SQLI pour mettre en œuvre DevOps.

L'entreprise vient de définir un modèle de maturité de livraison continue suivant les 5 thèmes [cf. Figure 9] :

- Culture et organisation
- Design et architecture
- Build et déploiement
- Test et vérification
- Information et reporting

De ce modèle ; l'entreprise déduit une vision globale de la livraison continue de valeur [cf. Figure 8] qui se base sur plusieurs pratiques et concepts clés :

- Infrastructure en tant que code (IaC) : C'est traiter la configuration d'infrastructure comme code.
- Pipeline en tant que code : C'est définir un pipeline comme code, au lieu de le configurer dans un outil d'intégration continue.

Nous n'avons cité que les concepts les plus importants pour notre projet de fin d'études.

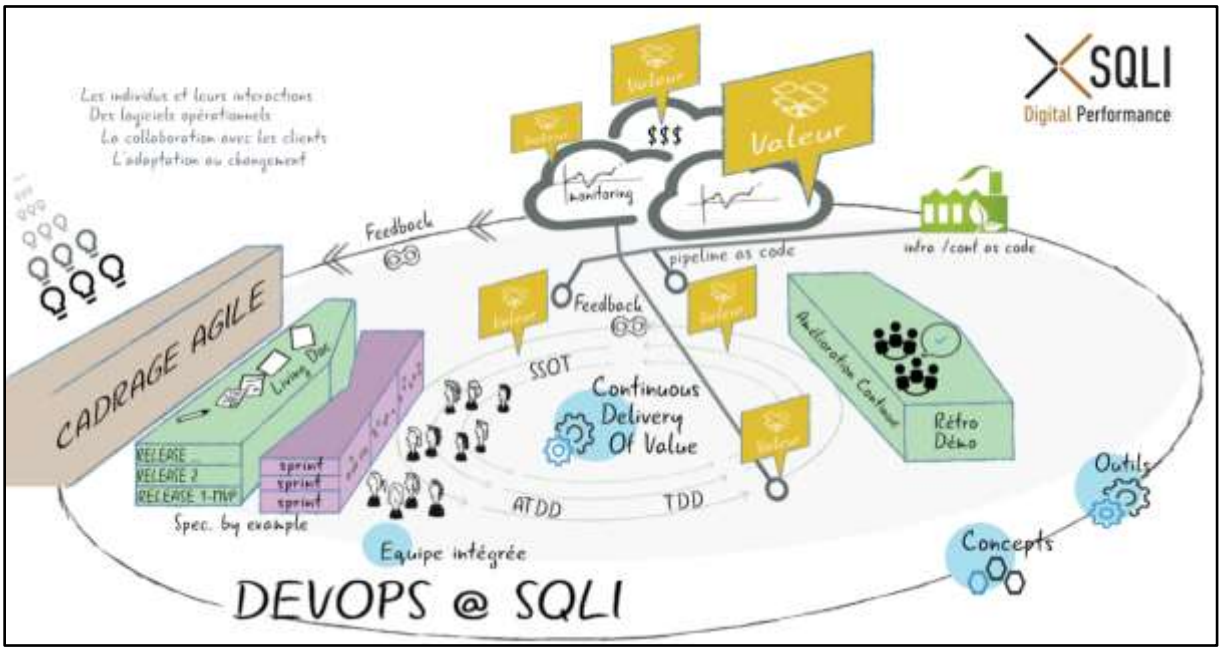


Figure. 8 : Vision DevOps chez SQLI

Pour terminer cette partie, nous citons que ce projet de fin d'études entre dans cette démarche et vise à mettre en œuvre le concept de l'infrastructure en tant que code et le pipeline en tant code chez SQLI.

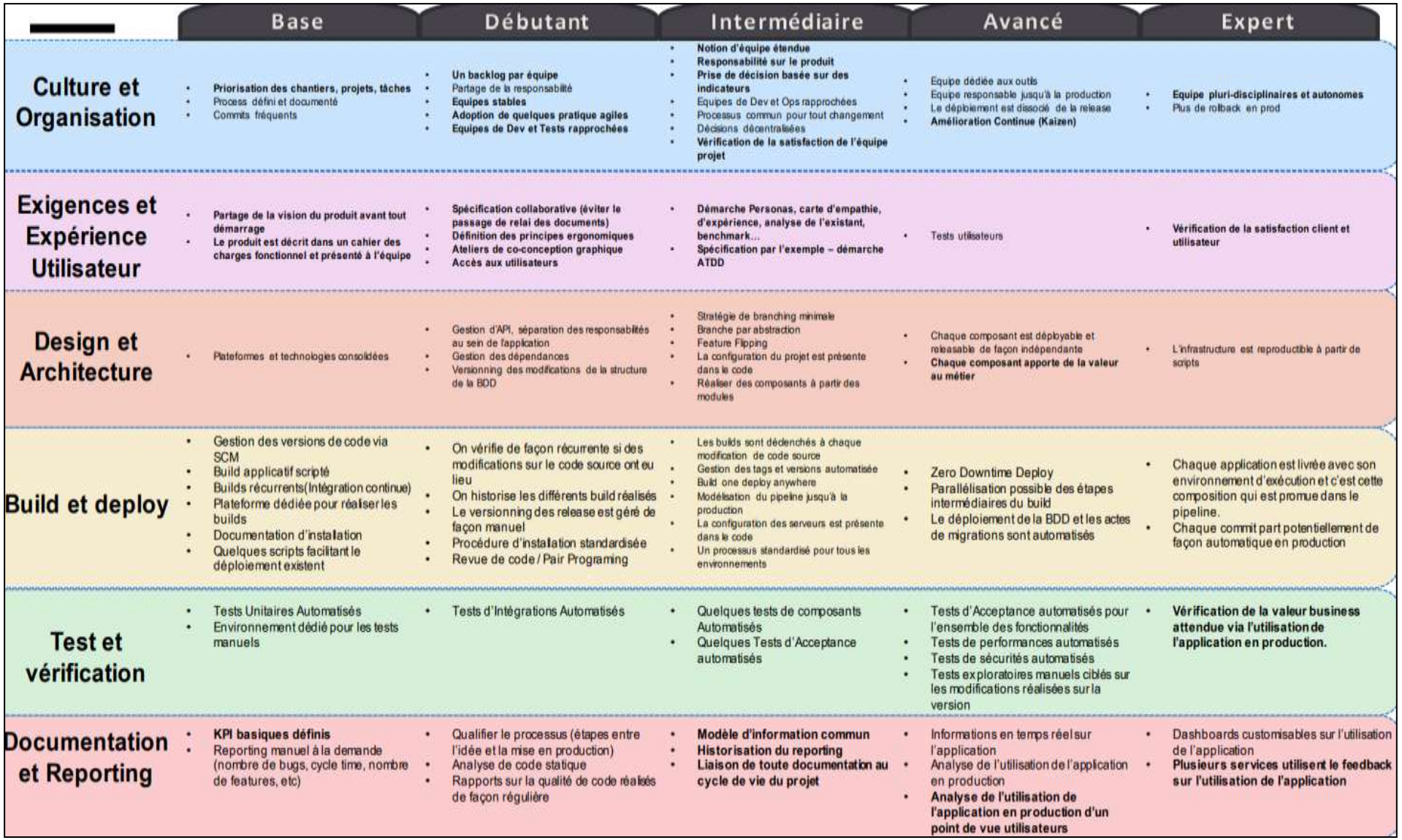


Figure. 9 : Modèle de maturité de livraison continue

3. Etude de l'existant

Dans Cette partie, nous présentons le pipeline de livraison continue existant chez SQLI pour le projet AMC, ainsi que nos constatations sur l'état actuel afin de donner une solution aux différents problèmes.

3.1. Fonctionnement actuel

Pour gérer le développement et la livraison de ses projets, l'entreprise SQLI dispose d'un pipeline d'intégration et livraison continue. Ce pipeline se constitue de [cf. Figure 10] :

- Gestionnaire de code source GitLab
- Serveur d'intégration continue Jenkins
- Gestionnaire de dépôt d'objets binaires Nexus
- Serveur de qualimétrie logicielle.

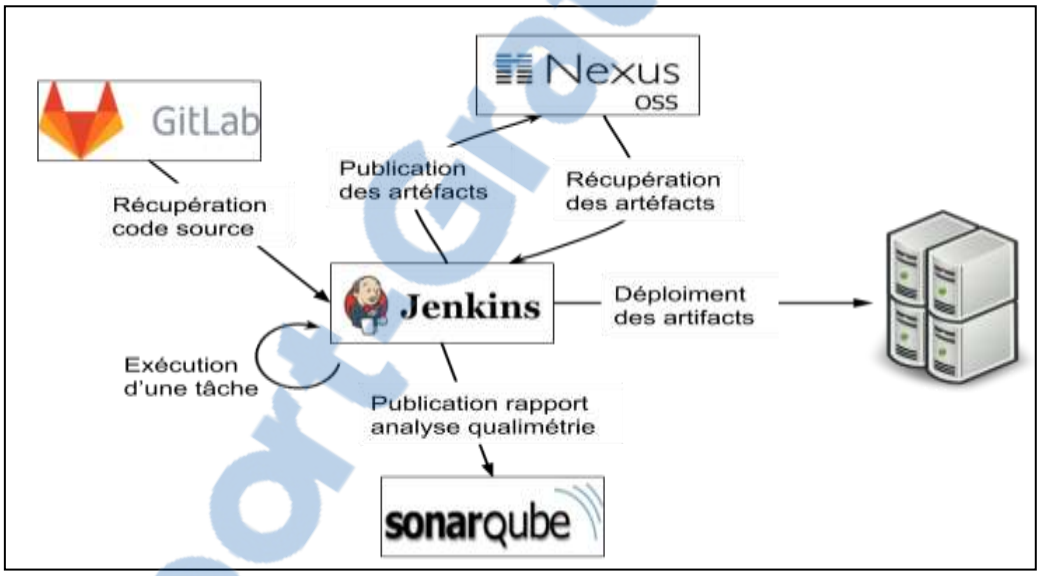


Figure. 10 : Interaction entre les composants du pipeline initiale

Dans cet état, le serveur Jenkins joue un rôle orchestrateur entre les autres outils à travers des tâches. Une tâche est une manière de compiler, tester, empaqueter, déployer ou d'effectuer des actions sur un projet.

Les tâches de Jenkins apparaissent sous plusieurs formes :

- Compilation et test unitaire d'un projet
- Création des rapports qualimétriques pour code source
- Empaquetage d'un projet pour une livraison
- Déploiement en environnement de production, test, ou développement.

Par exemple, le projet AMC sur lequel nous appliquons notre travail a les tâches suivantes [cf. Figure 11] :

- *AMC_Build* : Compilation et exécution des tests unitaires sur le projet après récupération du code source depuis GitLab.

- *AMC_Sonar* : Analyse de code source et publication du rapport d'analyse sur SonarQube.
- *AMC_Deploy* : Déploiement des artéfacts de projet dans les environnements (développement, intégration, Test, préparation, production) après leur récupération depuis Nexus.

Figure. 11 : Ensembles des tâches pour le projet AMC

De plus des environnements où sont installés les outils du pipeline, chaque projet dispose des environnements de déploiements suivants :

- Développement : un environnement où le projet est développé et où les tests unitaires sont performés (ordinateur du développeur).
- Intégration : le but de cet environnement est de combiner le travail d'une équipe de développeurs sur un projet pour le tester.
- Test : cet environnement permet aux testeurs de lancer les tests fonctionnels et les tests de régressions automatisées ou non sur le projet pour valider la qualité. Et de plus, il est utilisé pour reproduire les bogues.
- Préparation (Pre-production) : environnement similaire à l'environnement de production pour exécuter les tests d'acceptantes avant le déploiement en production ou pour exécuter des tests de performance.
- Production : l'environnement qui permet aux utilisateurs finaux d'interagir avec le projet.

3.2. Critique de l'existant

Suite à la vue globale présentée ci-dessus, nous avons constaté les problématiques suivantes que l'entreprise vise à résoudre :

- Les tâches Jenkins sont lancées manuellement par les membres d'équipe de projet au lieu d'être lancé automatiquement à chaque changement du code source.
- Tous les environnements sont manuellement configurés (perte du temps), ce qui ne permet pas un provisionnement rapide (allocation et configuration automatique) lors d'un événement critique, ou une demande d'un environnement similaire.
- La tâche Jenkins de déploiement se base sur un script Shell qui exécute des commandes sur l'environnement cible afin de déployer les artéfacts. Ce script présente plusieurs désavantages :
    - Difficile à tester et maintenir.
    - Ne supporte pas des stratégies de déploiement sans indisponibilité.
    - Impossibilité de gérer les informations sensitives sans divulgation.

4. Solution proposée

Pour remédier aux problématiques citées dans l'étude de l'existant, nous avons proposé après plusieurs discussions avec l'équipe DevOps de :

- Utiliser un gestionnaire de configuration, Ansible, pour gère la configuration et la préparation des environnements, ainsi que pour gérer le déploiement des artéfacts.
- Regrouper les différentes tâches Jenkins dans une seule tâche qui s'exécute à chaque modification du code source.

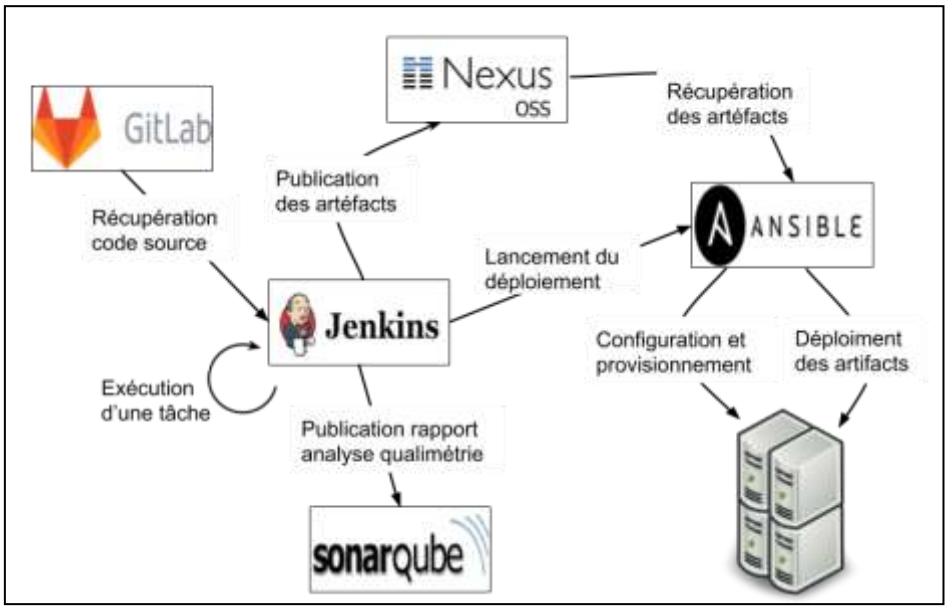


Figure. 12 : Interaction des composants du pipeline après intégration Ansible

L'amélioration du pipeline initial avec l'intégration de l'outil d'automatisation Ansible [cf. Figure 12] fournit plusieurs avantages :

- Améliorer et optimiser le processus de déploiement.
- Facilité de reprendre ou de créer une réplication d'un environnement dans le cas d'une panne d'infrastructure.

5. Démarche adoptée

Dans cette partie, nous exposons la démarche de travail adopté pour la gestion et la réalisation de notre projet.

5.1. Méthode adopté**e**

Afin d'assurer le bon déroulement de notre projet, nous avons opté pour la méthode agile Scrum, pour pouvoir répondre au mieux au besoin exprimé par l'entreprise en minimisant les risques de dépassement des délais dans le but d'arriver à une satisfaction partagée en préservant une bonne conduite du projet.

Scrum (« la mêlée ») est une méthode agile qui s'applique à la gestion de projet. Son principe est abordé pour la première fois dans un article de Hirotaka Takeuchi et Ikujiro Nonaka intitulé « The New New Product Development Game » publié en 1986 dans la Havard Business Review. Elle a ensuite été développée dans les années 1990 et formalisée par Ken Schwaber et Jeff Sutherland en 1995 [2].

Le principe de cette méthode est de focaliser l'équipe de manière itérative sur les fonctionnalités à réaliser. Le projet est ainsi découpé en modules fonctionnels qui seront réalisés, testés et livrés par séquences itératives appelées « sprint ». Chaque sprint vise à atteindre un but à partir duquel sont choisies les fonctionnalités à implémenter dans cette phase [cf. Figure 13].

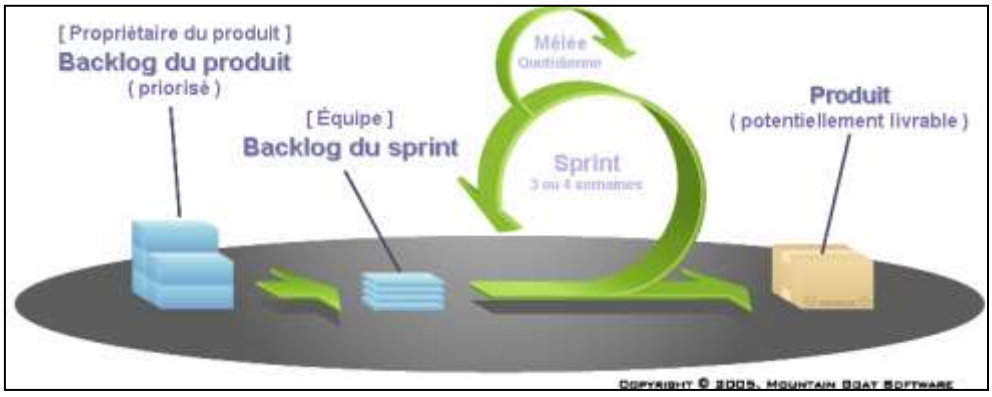


Figure. 13 : Vue globale du processus Scrum

Ainsi, nous avons adapté cette méthode à notre cas, en répartissant le projet sur des sprints d'une à trois semaines. À la fin de chacune, une réunion a été tenue pour présenter les tâches réalisées durant le sprint en question tout en fixant les tâches et objectifs à atteindre durant l'itération suivante.

Chaque fois qu'une différence est constatée pendant l'inspection, nous étions amenés à adapter le processus en question. Nous avons en fait, planifié des réunions hebdomadaires, en parallèle à ces Sprint-Reviews, pour présenter les actions de la période, les difficultés rencontrées ainsi que les actions de la période suivante et ses perspectives.

Ceci a permis de bien fixer les objectifs à chaque incrément et de pouvoir les adapter, mais aussi, de pouvoir à tout moment compléter ou modifier la liste des fonctionnalités à réaliser pour les prochains sprints.

5.2. Planification du projet

La planification du projet est une phase inéluctable pour le management de projet. Elle permet de définir les travaux à réaliser, fixer les objectifs, coordonner les actions, maitriser les moyens, diminuer les risques, suivre les actions en cours et rendre compte de l'état d'avancement du projet. Nous avons établi en premier lieu un découpage de projet en tâches,

puis un diagramme de Gantt qui permettront d'identifier les travaux à compléter en traduisant le projet en une liste de tâches à accomplir.

5.2.1. Découpage de projet en Sprint

Le projet s'étale sur 5 sprints, le titre de chaque sprint est une fonctionnalité de notre backlog du projet.

Numéro	Titre sprint	Description
1	Étude comparative des outils DevOps	Une étude qui vise à valider le choix, montrer les inconvénients et les avantages des outils déjà utilisés chez SQLI.
2	Mise en place du pipeline de livraison continue	Installer et configurer les outils choisis lors de l'étude comparative.
3	Gestion de configuration des environnements de projet AMC	Développement des « playbooks » et des « rôles » permettant de configurer et provisionner les machines des environnements AMC.
4	Gestion de configuration des environnements du pipeline	Développement des « playbooks » et des « rôles » Ansible permettant de configurer et provisionner les machines du pipeline.
5	Réaliser un pipeline en tant que code	Regrouper la totalité des tâches Jenkins dans un seul pipeline exécuté automatiquement.

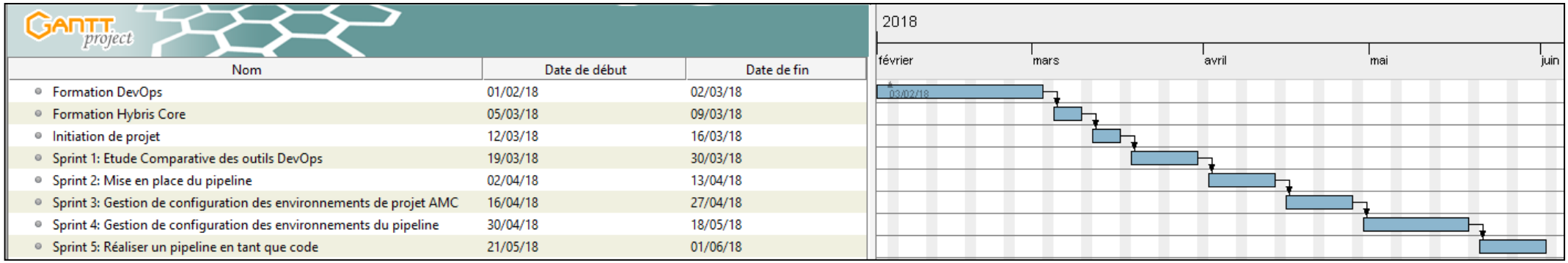
Table. 1 : Liste des sprints avec descriptions

5.2.2. Diagramme de Gant

Le diagramme de Gantt permet de visualiser les différentes tâches et leurs durées [cf. Figure 14], à citer qu'avant l'initiation de projet, nous avons effectué une formation d'un mois sur les outils DevOps et une autre sur la plateforme e-commerce Hybris.

Conclusion

Dans ce chapitre, nous avons présenté le cadre général du projet, en mettant en valeur les solutions des problématiques à résoudre, ainsi que la démarche adoptée pour la gestion de projet. Le chapitre suivant mettra plus de lumière sur les concepts clés de la livraison continue et la gestion de configuration.



Nom	Date de début	Date de fin
Formation DevOps	01/02/18	02/03/18
Formation Hybris Core	05/03/18	09/03/18
Initiation de projet	12/03/18	16/03/18
Sprint 1: Etude Comparative des outils DevOps	19/03/18	30/03/18
Sprint 2: Mise en place du pipeline	02/04/18	13/04/18
Sprint 3: Gestion de configuration des environnements de projet AMC	16/04/18	27/04/18
Sprint 4: Gestion de configuration des environnements du pipeline	30/04/18	18/05/18
Sprint 5: Réaliser un pipeline en tant que code	21/05/18	01/06/18

Figure. 14 : Diagramme de Gantt

Chapitre 2: Etude comparative des outils DevOps

Introduction

Pour mettre en place un pipeline de livraison continue, nous avons besoin d'un ensemble d'outils. Mais face à l'abondance des outils dans le marché, choisir la combinaison gagnante reste une tâche laborieuse. Pour cela, une étude comparative est nécessaire pour choisir les meilleurs outils répondant aux besoins techniques de notre projet.

L'outil approprié doit comprendre des fonctionnalités adéquates, mais aussi respecte les critères que nous avons étudiés. La démarche suivie et les détails l'étude effectuée sont présentés dans la suite de ce chapitre.

1. Démarche de l'étude comparative

Choisir une combinaison des outils DevOps homogène et ayant une communication automatique entre eux est un facteur majeur du succès d'un pipeline de livraison continue.

Plusieurs outils existent dans le marché. Le choix, donc, dépend des critères à prendre en considération.

Pour bien organiser cette étude, qui constitue une phase préalable à la phase de mise en œuvre du pipeline, nous suivons les étapes suivantes :

- Lister les outils les plus connus aux marchés et sélectionner les plus intéressants pour les analyser et étudier.
- Élaborer des critères de comparaison pertinents et rigoureux. De plus, nous ne tiendrons pas compte des critères non différenciâtes malgré leur importance.
- Comparer les outils selon les critères établis pour sortir avec des tableaux comparatifs, avant d'en faire la synthèse et identifier l'outil approprié.

Pour ce qui suit, nous allons présenter, pour chaque catégorie d'outils, l'étude comparative complète qui reflète exactement la démarche que nous venons de décrire.

2. Outils de gestion de code source

La gestion du code source (SCM) est la façon dont les changements logiciels sont effectués. Le SCM a un certain nombre d'objectifs qui portent fondamentalement sur la garantie que les équipes de développement puissent livrer des changements de code de qualité plus rapidement [3]. En améliorant le suivi, la visibilité, la collaboration et le contrôle à travers le cycle de vie des livraisons, les outils de SCM permettent plus de créativité, de liberté et offrent plus d'options aux développeurs travaillant sur des projets complexes. De plus, le SCM peut protéger les fichiers source de toute sorte d'anomalies, et permet à toutes les équipes de savoir qui a effectué quel changement et à quel stade.

Cette étude de gestionnaire de code source a pour but de consolider le choix de l'outil GitLab, qui est déjà utilisé en interne, face à ses concurrents BitBucket et GitHub.

2.1. Choix des critères

Les critères qui vont orienter notre choix d'outil de gestion de code source sont :

- Open Source : le code source du logiciel est accessible.
- Licence : gratuit ou payant.
- Gestionnaire de version : les gestionnaires de version supportés.
- Fonctionnalité intégration continue : possibilité de lancer des compilations et teste de code sans serveur d'intégration extérieur.
- Suivi de problèmes : Suivre les incidents et les anomalies d'un projet sans recourir à un autre outil.

2.2. Analyse

2.2.1. GitLab

GitLab est un gestionnaire web de référentiel git incluant un wiki et des fonctionnalités de suivi de problèmes. GitLab fournit une gestion centralisée des référentiels Git, permettant aux utilisateurs d'avoir le contrôle complet de leurs référentiels ou projets [3].

Écrit en Ruby (avec des compléments Go), le gestionnaire inclut contrôles d'accès granulaires, revues de code, suivi de problèmes, flux d'activités, wikis, et intégration continue. En décembre 2016, il regroupe 1400 contributeurs Open Source et est utilisé par les grandes entreprises comme Sony, IBM, CERN, NASA, etc.

2.2.2. GitHub

GitHub est un service de référentiel web hébergé offrant toutes les fonctionnalités de gestion de code source, tout en garantissant un espace aux développeurs pour stocker leurs projets et concevoir des logiciels en parallèle. GitHub fournit des fonctions de collaboration, de contrôle d'accès, des Wikis et des outils simples de gestion de tâches pour projets [3].

Conçu par des développeurs pour les développeurs, GitHub propose une interface graphique et un bureau web ainsi qu'une intégration mobile. Il ne se borne pas au développement logiciel : son aspect ouvert et « réseau social » est fondamental. Il permet de faire une copie du projet public d'une autre personne et de modifier ses fonctionnalités tout en visualisant le travail et les profils de chacun.

2.2.3. BitBucket

BitBucket est un service web hébergé pour projets logiciels utilisant le système de contrôle de version Mercurial ou git, et propose des plans commerciaux ou gratuits. Ces derniers incluent un nombre illimité de répertoires privés et jusqu'à 5 utilisateurs. Son élasticité simplifie la collaboration des équipes. L'extraction de requêtes, les permissions de branches et les discussions en ligne sont des fonctionnalités clés. Écrit en Python, en utilisant le Framework web Django, BitBucket permet aux équipes de livrer et de partager du code de meilleure qualité, plus rapidement. Utilisé pour son extrême élasticité, particulièrement dans

un environnement commercial, il garantit aux développeurs, vitesse et fiabilité quel que soit l'endroit où le projet [3].

2.3. Comparaison et synthèse

Le tableau synthétise la comparaison des gestionnaires de code source présentés préalablement, selon les critères établis et exposés auparavant.

Critère \ Outil	GitLab	GitHub	Bitbucket
Open Source	Oui	Non	Non
Licence	Gratuit pour la version communauté	Payant	Payant
Gestionnaire de version	Git	Git	Git, Mercurial
Fonctionnalité intégration continue	Intégré ou à travers d'autres outils	À travers d'autres outils seulement	À travers d'autres outils seulement
Suivi de problèmes	Oui	Oui	À travers un autre outil Jira

Table. 2 : Récapitulatif de comparaison des outils de gestion de code source

L'outil GitLab est open source, de plus gratuit pour la version communauté. Cette raison nous a poussées de le choisir sans hésitation. D'une autre, son intégration de fonctionnalité d'intégration continue et suivi de problèmes permettra dans le futur d'abonder l'utilisation des outils à part.

3. Outils d'intégration continue

Les logiciels d'intégration continue (CI) permettent aux développeurs de faire un commit de code vers un plus grand référentiel, aussi souvent qu'ils le souhaitent. Les outils construisent et testent le code de façon à ce que toute erreur ou tout bug soit détecté rapidement et transféré au développeur pour résolution [4]. Enfin, le CI facilite le processus de livraison logicielle, raccourcissant les cycles de livraison et laissant aux développeurs plus de liberté pour se concentrer sur l'innovation. Il permet à différents développeurs ou équipes de travailler en parallèle sur différents aspects du même projet.

Parmi les logiciels d'intégration continue existants sur le marché, nous avons choisi de comparer Jenkins le plus connu, Travis CI et CircleCI.

3.1. Choix des critères

Les critères qui vont orienter notre choix d'outil d'intégration continue sont :

- Open Source : le code source du logiciel est accessible.
- Licence : gratuit ou payant.
- Installation : Complexité de mettre en place l'outil pour un environnement de production.
- Gestionnaire de code source : supporté par défaut ou avec des plug-ins
- Système d'exploitation : supporté pour lancer des compilations et des tests de code.

3.2. Analyse

3.2.1. Jenkins

Jenkins est un outil d'intégration continu Open Source écrit en Java. Jenkins est un successeur de Hudson. Il supporte les outils SCM tels que Subversion, git, etc. Jenkins peut également exécuter des scripts Shell et des projets Ant ou Maven [4].

Jenkins dispose en outre de nombreux plug-ins qui le rendent compatible avec tous les langages de programmation et une grande majorité de systèmes de contrôle de version et de référentiels. Jenkins permet aux utilisateurs de concevoir et livrer des applications à grande échelle rapidement et supporte conception, déploiement et automatisation dans la plupart des projets.

3.2.2. TravisCI

Travis CI est un service hébergé d'intégration continue (IC) en Open Source pour concevoir et tester des projets hébergés sur GitHub. Les exécutions de builds et de tests sont déclenchées automatiquement toutes les fois qu'un commit est réalisé et poussées vers un référentiel GitHub [4].

Travis CI se configure en plaçant un fichier travis.yml dans le répertoire racine de votre référentiel. Travis CI a été conçu pour exécuter tests et déploiements en laissant les développeurs se concentrer sur le code. Cette automatisation facilite le déploiement simple, rapide et agile pour les équipes logicielles.

Travis CI est gratuit pour les projets Open Source, payant pour les projets commerciaux ou privés.

3.2.3. CircleCI

CircleCI est une plateforme d'intégration et de déploiement continus qui automatise les processus de build, de test et de déploiement. Il permet aux équipes de développement de déployer des projets logiciels rapidement, tout en facilitant l'élasticité [4].

CircleCI est un serveur Cloud hébergé qui réduit considérablement l'effort de test. Il supporte un grand nombre de technologies, comme Ruby on Rails, Sinatra, Node, Python, PHP, Java et Clojure.

Il dispose de quatre fonctionnalités : une configuration rapide, une intégration avec un grand nombre d'outils (conférant de la souplesse dans l'environnement de travail utilisateur), il supporte tous les tests et permet de configurer facilement un enchaînement. L'association de ces fonctionnalités permet à l'utilisateur de livrer des projets mieux testés plus rapidement.

3.3. Comparaison et synthèse

Le tableau synthétise la comparaison des serveurs d'intégration continue présentés préalablement, selon les critères établis et exposés auparavant.

Critère \ Outil	Jenkins	Travis CI	circleci
Open Source	Oui	Oui	Non
Licence	Gratuit	Gratuit, il existe une version payante	Payant
Installation	Facile	Difficile	Aucune installation, SaaS
Gestionnaire de code source	GitLab, GitHub, Bitbucket, TFS, CVS, Preforce	GitHub, Bitbucket	GitHub, Bitbucket
Système d'exploitation	Linux, MacOs, Windows, Unix	Linux, MacOs	Linux, MacOs

Table. 3 : Récapitulatif de comparaison des outils d'intégration continue

D'après le tableau de cette étude et les différentes informations collectées, nous sommes arrivés à la conclusion suivante :

Jenkins est l'outil le plus adapté pour notre cas, il est populaire et bénéficie d'une vaste communauté qui délivrent des plug-ins ouvrant un large champ de possibilité, contrairement à Travis CI, sans oublier son aspect open source et licence gratuit qui lui donne un pas d'avance par rapport à CircleCI.

4. Outils de qualimétrie logiciels

Les outils de Qualimétrie servent à mesurer la qualité de code, la couverture de tests, etc. Ainsi que publier des rapports sur les différents indicateurs obtenus. Une analyse correcte des données et des rapports publiés par la Qualimétrie permettra de mesurer de manière

objective la qualité du code et ainsi appréhender la dette technique qui s'accumule et anticiper les bugs à venir [5].

Nous avons tenu de comparer SonarQube le plus complet à ses nouveaux concurrents Scrutinizer et Codacy.

4.1. Choix des critères

Les critères qui vont orienter notre choix d'outil qualimétrie sont :

- Open Source : le code source du logiciel est accessible.
- Licence : gratuit ou payant.
- Nombre de langages supportés : nombre de langage que l'outil peut scanner et analyser.
- Communauté de support et documentation.

4.2. Analyse

4.2.1. SonarQube

SonarQube (précédemment appelé Sonar) est un logiciel libre développée par SonarSource permettant de mesurer la qualité du code source en continu, en effectuant une analyse statique du code pour détecter les bugs, les odeurs de code et les failles de sécurité sur plus de 20 langages de programmation [6].

SonarQube propose des rapports sur le code dupliqué, les normes de codage, les tests unitaires, la couverture de code, la complexité du code, les commentaires, les bogues et les failles de sécurité.

SonarQube peut enregistrer l'historique des métriques et fournir des graphiques d'évolution. Aussi, il fournit une analyse et une intégration entièrement automatisées avec Maven, Ant, Gradle, MSBuild et des outils d'intégration continue (Atlassian Bamboo, Jenkins, Hudson, etc.).

4.2.2. Scrutinizer

Scrutinizer est une plate-forme d'inspection continue qui aide à créer de meilleurs logiciels. En permettant de mesurer et suivre en continu la qualité du code avec des indicateurs de code simplifiés et faciles à comprendre. De plus, Scrutinizer offre la possibilité d'avoir des commentaires sur les changements de qualité du code entre les versions et recevoir des rapports hebdomadaires sur la qualité du code [7].

4.2.3. Codacy

Codacy est un outil automatisé d'analyse de qualité de code qui aide les développeurs à expédier de meilleurs logiciels, plus rapidement. Avec Codacy, vous obtenez des modifications d'analyse statique, de complexité cyclomatique, de duplication et de couverture de test des unités de code à chaque modification ou ajout de nouveau code.

Codacy peut appliquer des normes de qualité de code ce qui permet de gagner du temps dans les révisions de code et appliquer plus rapidement les meilleures pratiques de sécurité et de codage [8].

4.3. Comparaison et synthèse

Le tableau synthétise la comparaison des outils de Qualimétrie logiciel présentés préalablement, selon les critères établis et exposés auparavant.

Critère / Outil	sonarqube	scrutinizer	CODACY
Open Source	Oui	Non	Non
Licence	Gratuit	Payant	Payant
Nombre de langage supporté	20	7	11
Communauté de support et Documentation	Communauté large et documentation claire	Communauté assez pauvre, mais bien documenté	Communauté pauvre, mais une documentation riche

Table. 4 : Récapitulatif de comparaison des outils de qualimétrie logicielle

Dans cette étude SonarQube à un grand pas d'avance par rapport à ses concurrents Codacy et Scrutinizer. D'une part, pour son aspect open source et gratuité de licence. D'autre part, pour le grand nombre de langages qu'il supporte et sa large communauté.

5. Outils de gestion de dépôt d'objets binaires

Un gestionnaire de dépôts d'objets binaires (binary repository manager en anglais) est un outil logiciel conçu pour optimiser le téléchargement et le stockage des objets binaires utilisés et produits dans le cadre du développement de logiciels (généralement des fichiers). Il centralise la gestion de tous les artefacts générés et utilisés par les logiciels de l'organisation (éditeur logiciel). Il permet de répondre à la problématique de complexité découlant de la diversité des types d'objets binaires, de leur position dans l'ensemble du flux de travail, ainsi que les dépendances entre eux [9].

Un dépôt d'objets binaires est un dépôt de logiciels pour les paquets, artefacts et leurs métadonnées. Il peut être utilisé pour stocker les fichiers binaires produits par l'organisation elle-même (releases et builds intermédiaires des produits) ou pour stocker les binaires tiers (dépendances) qui doivent être traités différemment pour des raisons techniques et juridiques.

5.1. Choix des critères

Les critères qui vont orienter notre choix d'outil de gestion de dépôt d'objets binaires sont :

- Configuration à travers interface : possibilité de configurer l'outil à travers l'interface graphique.
- Format des composants supportés : les formats qui peut enregistrer et gérer.
- Stockage d'Artefact : méthode d'enregistrement des données dans le disque dur.

5.2. Analyse

5.2.1. Nexus

Nexus est un gestionnaire de référentiel qui organise, stocke et distribue des composants logiciels. Il est conçu pour standardiser tous les nouveaux systèmes et bibliothèques, permettant aux développeurs et aux équipes d'y accéder facilement. Il facilite par ailleurs la distribution logicielle [9].

Nexus offre de nombreuses fonctionnalités, et supporte tous les formats de composants connus. Il s'agit d'un produit en haute disponibilité, conçu en priorité pour la livraison et le déploiement continus. Il est également capable de détecter immédiatement et automatiquement toute faille de sécurité et des problèmes potentiels de licences.

Il fait partie de la suite des produits Nexus, conçus pour l'automatisation et l'élasticité de DevOps, et s'intègre à une grande variété de technologies.

5.2.2. Archiva

Apache Archiva est un logiciel de gestion extensible de référentiel, conçu pour la gestion des référentiels de composants personnels ou professionnels [9].

Il assure la gestion de proxy des référentiels distants, la gestion d'accès sécurisée, le stockage de composants de build, la livraison, la navigation, l'indexation, et offre des fonctionnalités extensibles de scan et de reporting.

5.2.3. Artifactory

JFrog Artifactory est un gestionnaire de référentiel de composants, indépendant de toute technologie, qui supporte les logiciels conçus avec tout type de langage ou tout type d'outil. C'est également le seul gestionnaire de référentiel professionnel qui supporte les référentiels Docker sécurisé, clusterisé et haute disponibilité [9].

Conçu pour s'intégrer avec la majorité des outils d'intégration et de livraison continue, JFrog Artifactory fournit une solution automatisée de bout en bout pour le suivi des composants, du développement à la production.

JFrog Artifactory est conçu pour les équipes de développement et de DevOps. Il sert de point d'accès unique qui organise toutes les ressources et élimine les complications. Il permet

également aux équipes de productions de faire transiter efficacement le flux continuel de code provenant de chaque PC de développeur vers l'environnement de production.

5.3. Comparaison et synthèse

Le tableau synthétise la comparaison des gestionnaires de dépôt d'objets binaires présentés préalablement, selon les critères établis et exposés auparavant.

Critère \ Outil	Nexus oss	archiva	artifactory
Configuration à travers interface	Totalement à travers admin web console	Partielle	À travers l'interface et l'API REST
Format des Composant supporté	Java, Docker, Npm, Bower, NuGet	Java seulement	Java seulement
Stockage d'Artefact	Stockage blob basé sur système de fichier	Système de fichier	Système de fichier

Table. 5 : Récapitulatif de comparaison des outils de gestion de dépôt d'objets binaires

Notre choix est fait pour Nexus, car il supporte dans sa version gratuit le stockage des images docker. En plus, il utilise une technique de stockage plus performante et offre une interface de configuration conviviale.

6. Outils de gestion de configuration et automatisation

Les outils de gestion de configuration sont utilisés pour contrôler les changements, les mises à jour et les modifications sur l'infrastructure d'une entreprise. Les outils gèrent l'équilibre souhaité au sein de grands environnements composés de serveurs, effectuent une supervision en continu, et assurent que l'infrastructure est configurée selon les bonnes spécifications, éliminant les risques et réduisant le temps de résolution des incidents. Ces fonctionnalités facilitent la collaboration à travers l'entreprise en éliminant les problèmes de contrôle de version et en garantissant l'uniformité des modifications. Enfin, les outils de gestion de configuration sont indispensables pour éviter les trop grandes variations entre serveurs et infrastructure, économisant de facto temps et coûts requis pour réparer les erreurs [10].

Dans la catégorie des outils de gestion de configuration trois outils sont les plus connus et utilisés par les entreprises : Ansible, Puppet et Chef.

Cette étude vient pour montrer les avantages et les inconvénients de l'outil Ansible qui est proposé pour ce projet face à Puppet et Chef.

6.1. Choix des critères

Les critères qui vont orienter notre choix d'outil de gestion de configuration et automatisation sont :

- Sans Agent : Pouvoir contrôler des machines sans configuration apriori ou installation d'agent.
- Langage
- Dépendance client : les logiciels pré requis pour configurer une machine.
- Mécanisme : de partage de configuration, push ou pull.
- Approche de description d'état de machine, déclaratif ou procédural.
- Installation : Complexité de mettre en place l'outil pour un environnement de production.
- Contributeur : nombre de contributeur dans le code source de l'outil.

6.2. Analyse

6.2.1. Ansible

Ansible est un moteur d'automatisation Open Source qui automatise le provisionnement logiciel, la gestion de configuration et le déploiement applicatif [10].

Ansible livre une automatisation IT simple, qui met fin aux tâches répétitives et libère les équipes DevOps pour qu'elles se concentrent sur des tâches plus stratégiques. Ansible facilite le déploiement applicatif, la gestion de configuration et l'orchestration, le tout depuis un système unique.

Pour y parvenir, Ansible a été conçu comme un outil simple, fiable, et convivial, contenant un minimum de dépendances. De plus, son architecture sans agents garantit sa sécurité et réduit les frais de réseau.

6.2.2. Chef

Chef est un outil de gestion de configuration et une plateforme d'automatisation conçu pour rationaliser les tâches de provisionnement, de configuration et de maintenance des serveurs d'une entreprise. Il transforme l'infrastructure en code, la rendant souple, versionable, lisible et testable, quelle que soit la plateforme où il s'exécute ou la taille du réseau [10].

Écrit en Ruby et Erlang, il peut s'intégrer avec une grande variété de plateformes cloud, y compris Rackspace, Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, et Microsoft Azure.

Comme son nom l'indique, Chef requiert l'écriture d'une série de recettes décrivant une série de ressources qui doivent se trouver dans un état particulier : les packages devant être installés, les services devant être exécutés, ou les fichiers devant être écrits.

6.2.3. Puppet

Puppet est un outil de gestion de configuration logicielle Open Source qui assure un moyen standard de livrer et d'exploiter les logiciels, quel que soit l'endroit où ils s'exécutent. L'utilisateur doit déclarer le statut final des applications déployées et de l'infrastructure provisionnée au moyen d'un langage facile d'accès [10].

Compatible avec Unix et Windows, Puppet est livré avec son propre langage déclaratif pour décrire la configuration du système. Il existe deux éditions de l'outil : Open Source et professionnelle. Cette dernière inclut un GUI, des API et des outils en ligne de commande pour la gestion des nœuds.

L'objectif de Puppet est de permettre de partager, de tester et d'appliquer des changements à travers un Datacenter, tout en garantissant visibilité et reporting pour la prise de décision et la conformité. Son véritable objectif est de mettre en œuvre une manière élastique et standard d'automatiser le déploiement et la mise en production des applications.

6.3. Comparaison et synthèse

Le tableau synthétise la comparaison des gestionnaires de configuration présentés préalablement, selon les critères établis et exposés auparavant.

Critère \ Outil	ANSIBLE	CHEF	puppet
Sans Agent	Oui	Non	Non
Langage	Python	Ruby	Ruby
Dépendance client	Python, sshd, bash	Ruby, sshd, bash	Ruby
Mécanisme	Push	Pull	Pull
Approche	Procédural	Procédural	Déclarative
Installation	Facile	Peu facile	Difficile
Contributeur	3432	522	492

Table. 6 : Récapitulatif de comparaison des outils de gestion de configuration et automatisation

L'avantage d'Ansible par rapport aux autres outils de gestion de configuration est son aspect sans agent, c'est-à-dire, il n'a pas besoin d'une configuration a priori pour contrôler une machine. De même, sa communauté est plus grande, et chaque mois de nouvelles options sont ajoutées.

Donc, Ansible reste le plus adapté et facile à mettre en œuvre en comparaison avec Chef et Puppet.

7. Synthèse et choix

À l'issu de chaque étude comparative, nous avons défini notre choix technologique parmi les outils énoncés, mais nous allons récapituler.

Comme gestionnaire de code source nous avons opté pour GitLab, et pour l'outil de qualimétrie, note choix c'est porté sur SonarQube.

Pour la gestion de configuration, nous avons utilisé Ansible. Et comme serveur d'intégration continue Jenkins. Et pour le dépôt des artéfacts, nous avons gardé Nexus. Le tableau récapitule nos choix.

Fonctionnalité	Outil
Gestionnaire de code source	GitLab
Serveur d'intégration continue	Jenkins
Outil de qualimétrie logiciel	SonarQube
Dépôt d'objet binaire	Nexus
Gestionnaire de configuration	Ansible

Table. 7 : Tableau récapitulatif des outils choisis

Conclusion

Au cours de ce chapitre, nous avons fait une étude comparative des outils à mettre en place dans notre pipeline de livraison continue. Dans un premier temps, nous avons défini la démarchée à suivre, ensuite, nous avons comparé chaque catégorie d'outils pour sortir avec notre choix. Enfin, nous avons fait une synthèse des outils choisis.

Chapitre 3: Analyse et Conception

Introduction

À travers ce chapitre, nous entamons l'analyse et la conception qui présente une étape fondamentale qui précède la réalisation. Nous commençons par une description des objectifs du projet et des besoins. Nous procédons ensuite à la définition des acteurs et un cas d'utilisation global du pipeline. Et nous terminons par représenter l'infrastructure du système et l'interaction entre ces composants.

1. Objectifs du projet

Les objectifs de ce projet sont :

- Mettre en place un pipeline de livraison continue qui supporte la gestion de configuration.
- Mettre en œuvre la pratique du pipeline en tant que code, en anglais « Pipeline as Code ».
- Mettre en œuvre la pratique de l'infrastructure en tant que code, en anglais « Infrastructure as Code ».

Les deux derniers objectifs seront appliqués sur le projet AMC, dans une vision d'étaler ces pratiques sur d'autres projets.

2. Analyse des besoins

Dans cette partie, nous commençons par analyser les besoins fondamentaux de notre système à travers la partie besoins fonctionnels. Puis, nous finissions par les besoins non fonctionnels à considérer pour ce système.

2.1. Besoins fonctionnels

Un recensement a permis de dévoiler les besoins suivants :

- Lancement automatique des tâches Jenkins suite à une requête de fusion GitLab.
- Notification d'utilisateur en cas d'échoue d'une tâche Jenkins.
- Supporter la compilation et la construction de projet avec des systèmes d'exploitation différents.
- Authentification centralisée des utilisateurs à travers LDAP.
- Automatisation de déploiement avec Ansible.
- Configuration des environnements avec Ansible.
- Possibilité de livrer n'importe quand.

2.2. Besoins non fonctionnels

De plus des besoins fonctionnels déjà cités, notre système de livraison continue doit respecter un nombre de critères donnant une meilleure qualité de la solution obtenu :

- Performance : le temps de réponse du système (les différents outils) doit être minimal.

- Extensibilité : le système doit pouvoir supporter l'évolution et l'extensibilité de ses composants, possibilité d'ajouter des nœuds en cas de montée en charge.
- Convivialité : les interfaces des outils doivent être intuitives et faciles à comprendre, et permettre aux utilisateurs de réaliser leurs objectifs sans ambigüité.

3. Identification des acteurs

Pour une plateforme de livraison continue, nous identifions quatre acteurs :

- Développeur : c'est utilisateur qui peut modifier le code source pour un objectif donné (ajout d'une fonctionnalité, correction d'erreur, etc.) et lancer après des tâches Jenkins pour assurer l'intégration continue et la livraison continue du projet.
- Testeur : c'est un utilisateur qui lance des tâches Jenkins (normalement de déploiement dans les environnements de test) afin d'assurer la régression et la conformité des changements aux besoins ou pour tester les fonctionnalités du projet.
- Chef de projet : c'est un utilisateur qui hérite du développeur, mais il fait aussi le suivi des correctifs et la réalisation des fonctionnalités.
- Configurateur : c'est un utilisateur qui peut configurer les environnements d'un projet d'une manière manuelle ou automatique avec Ansible. De plus, il est l'administrateur de la plateforme de livraison continue.

4. Cas d'utilisation global du pipeline

Nous voulons donner une version globale du comportement fonctionnel de notre plateforme de livraison continue. Le diagramme de cas d'utilisation ci-dessous liste les cas d'utilisations généraux, ainsi que les acteurs interagissant avec le système.

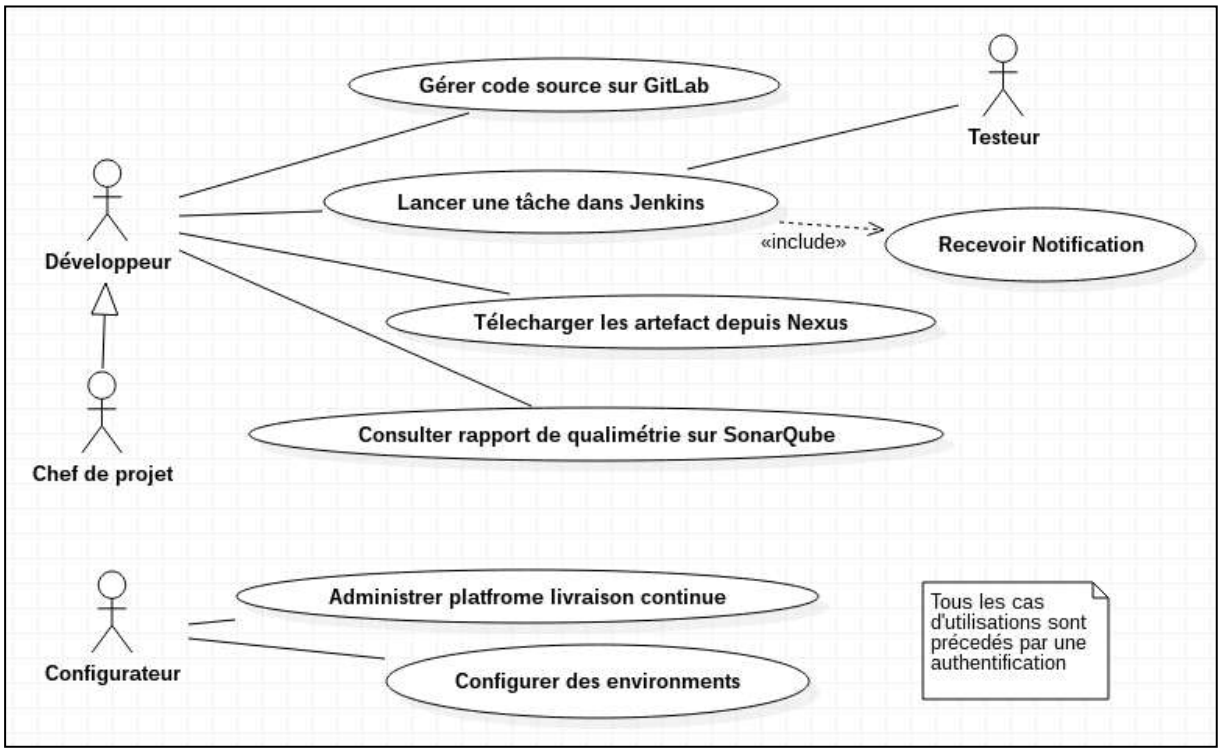


Figure. 15 : Diagramme de cas d'utilisation général

Pour lever l'ambigüité et rendre clairs les cas d'utilisation, nous donnons une description de chaque cas dans le tableau suivant :

Cas d'utilisation	Description
Gérer code source sur GitLab	Le développeur peut récupère le code source, et le pousser sur GitLab après modification.
Lancer une tâche dans Jenkins	Le développeur et le testeur peuvent lancer depuis Jenkins plusieurs tâches : - Compilation et exécution des tests unitaire - Analyse de qualimétrie et publication du rapport - Publication des Artéfacts - Déploiement dans un environnement - Annuler une tâche en cours d'exécution - Ce cas inclut la notification de l'acteur en cas d'erreur.
Télécharger les artefacts depuis Nexus	Le téléchargement des artéfacts publié récemment pour les tester par les développeurs ou les testeurs.
Consulter rapport de qualimétrie sur SonarQube	La consultation du rapport de qualimétrie permet au développeur et au chef de projet d'examiner les métriques de qualité logicielle.
Administre plateforme livraison continue	L'administration des outils du pipeline de livraison continue, contient plusieurs actions : - Gérer les utilisateurs et leurs droits d'accès - Configurer les outils (Authentification LDAP, Notification par mail, etc.) - Gérer les dépôts code source GitLab (Création, Configuration, Suppression) - Gérer les tâches Jenkins (Création, Configuration, Suppression) - Gérer les dépôts d'objet binaire Nexus - Gérer les bornes de qualité (Quality Gate) et les profiles de qualité sur SonarQube.
Configurer environnements avec Ansible	La configuration des environnements est une étape essentielle avant le déploiement des artéfacts. Avec Ansible, nous créons des « playbooks », leur exécution permet de mettre en places toutes les exigences d'un artéfact.

Table. 8 : Description des cas d'utilisations

Après cette description, nous rappelons et il est clair d'ailleurs que nous avons essayé de donner une vue générale de l'interaction des acteurs avec notre système sans entrer dans les détails et essayer de recenser tous les cas d'utilisation possibles.

5. Architecture physique globale du pipeline

Avant de mettre en place et implémenter le pipeline de livraison continue, nous devons estimer les ressources informatiques nécessaires à allouer. Le diagramme de déploiement ci-dessous donne une vue globale des nœuds physiques du système et leurs relations.

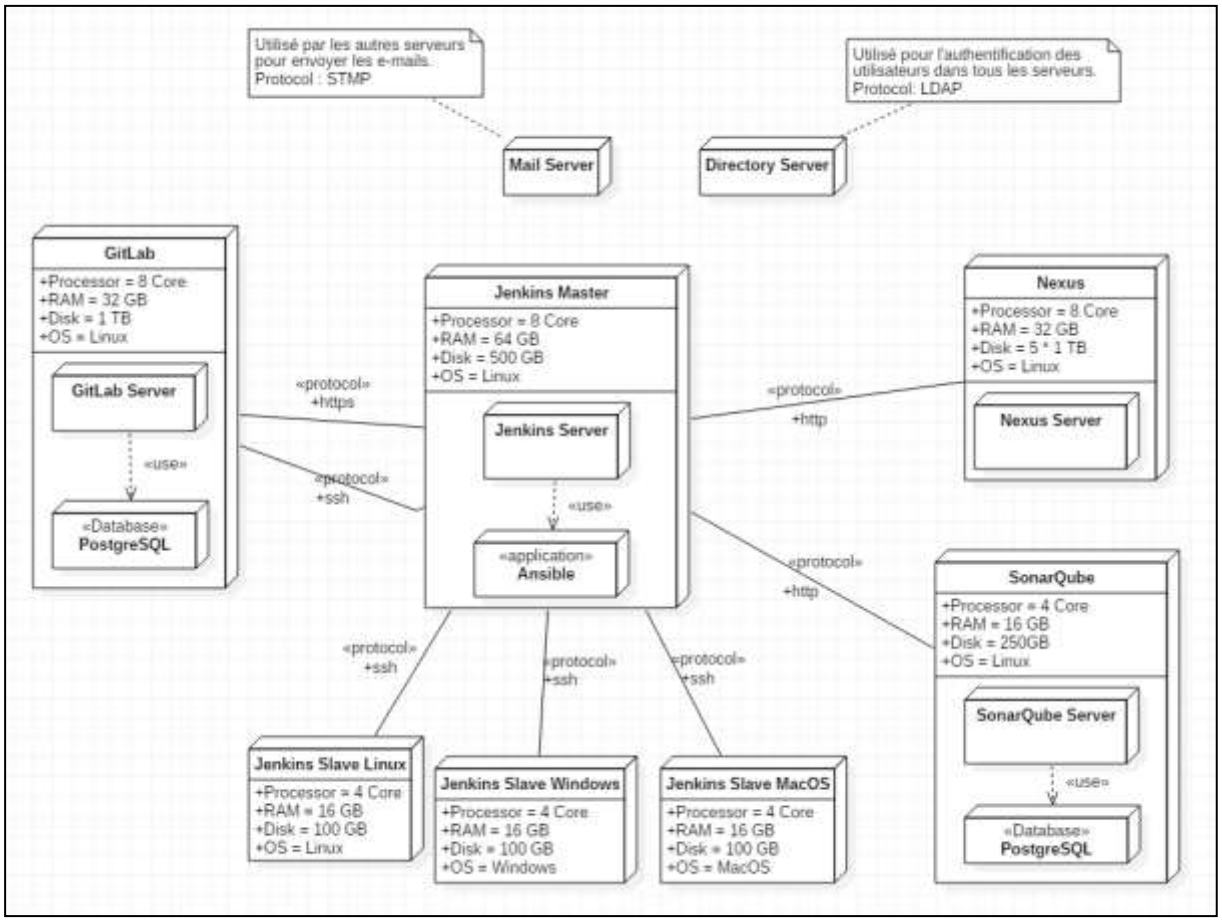


Figure. 16 : Diagramme déploiement pipeline livraison continue

Dans le diagramme, nous remarquons que le serveur Jenkins a plusieurs esclaves. Ces esclaves sont dédiés pour lancer des tâches de compilation et de construction des projets. De cette manière, nous pouvons construire des projets sur des systèmes d'exploitation différents (Linux, Windows, MacOs). Et nous pouvons étendre notre système en cas d'une monté en charge sur les esclaves par une simple réplication d'un nœud esclave.

Nous ajoutons que le nœud serveur de messagerie et serveur d'annuaire ont une communication avec tous les autres nœuds. Le premier pour assurer la notification par messagerie électronique lors des événements déclenchants (échec, erreur, succès, espace insuffisant, etc.). Le deuxième permet de fournir une authentification centralisée des utilisateurs aux différents outils. Les détails des ressources (CPU, RAM, Disc) de chaque nœud sont déduits depuis la documentation officielle de chaque outil en tenant compte de quelques critères (Nombre d'utilisateurs estimé, etc.).

6. Architecture physique des environnements du projet AMC

Le projet AMC dont nous avons déjà cité [cf. Paragraphe 1.1.5] se base sur la plateforme e-commerce Hybris. Afin de livrer en continu et avec qualité, le projet à plusieurs environnements (intégration, test, préparation, production).

Après que nous donnons un aperçu de l'architecture générale typique de l'environnement, SAP Hybris commerce. Nous donnerons l'architecture physique spécifique de chaque environnement du projet AMC.

6.1. Architecture Déploiement Hybris

Le diagramme suivant illustre une architecture de déploiement qui mappe les composants clés sur une infrastructure commune [11]. Ce n'est pas un plan exact qui peut être utilisé pour n'importe quel projet, mais qui donne une idée juste des composants matériels typiques utilisés dans l'architecture de déploiement.

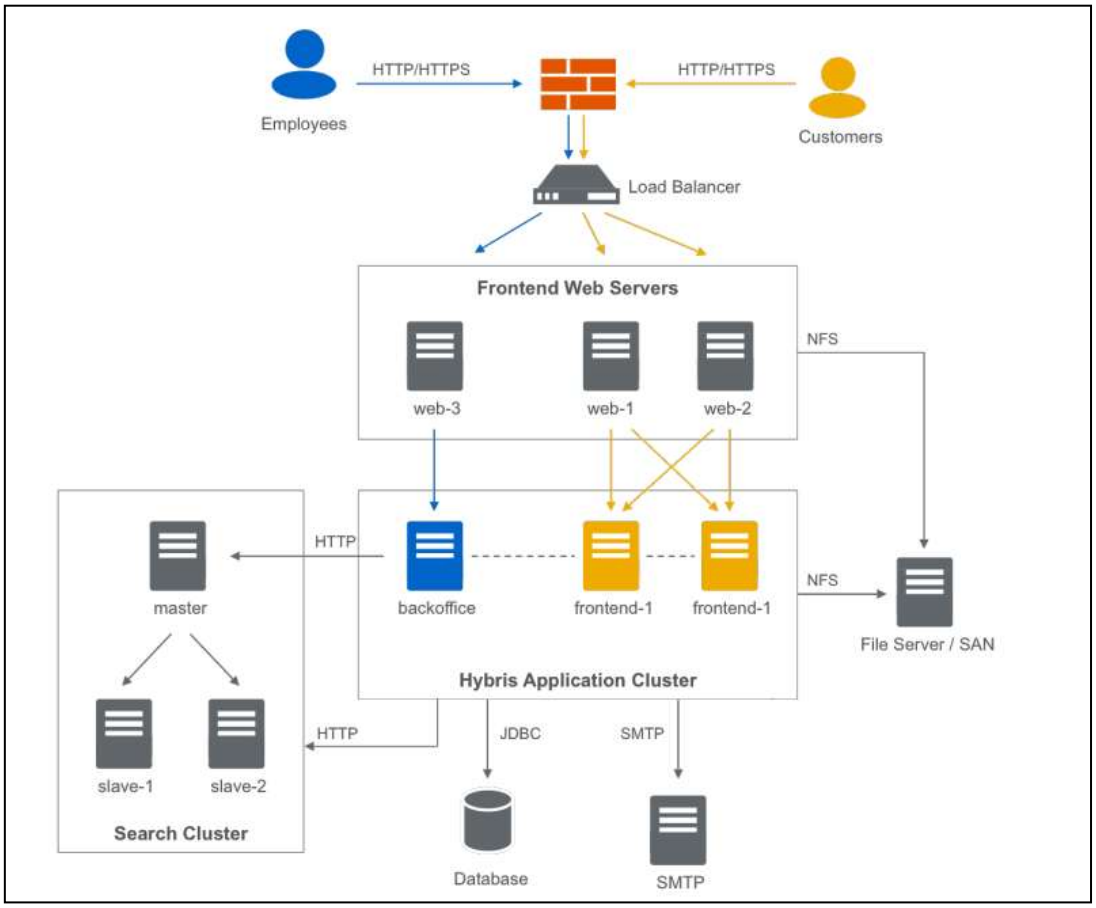


Figure. 17 : Architecture de déploiement commun

Plus de détails sur chaque composant sont donnés dans le tableau ci-dessous.

Composant	Rôle
Équilibreur de charge	L'utilisation principale d'un équilibreur de charge consiste à équilibrer le trafic sur plusieurs serveurs afin que la charge puisse être partagée entre plusieurs nœuds. Cela améliorera également la fiabilité du système, dans le cas d'une panne d'un nœud, l'équilibreur de charge peut acheminer les demandes entrantes vers d'autres nœuds.
Serveur web	Les serveurs Web agissent comme un proxy inverse pour transférer les demandes entrantes. Les serveurs Web aideront à sécuriser l'application, à accélérer le trafic Web et à réduire la charge sur les serveurs d'applications.
Cluster d'application Hybris	Le cluster Hybris est constitué de plusieurs serveurs d'applications Hybris Commerce, souvent appelées nœuds. Un cluster doit séparer le trafic client des traitements en ajoutant des nœuds de Frontend et de back-office spécifique.
Serveurs de recherche	Pour des raisons fonctionnelles et de performance, il est fortement recommandé d'utiliser une technologie de recherche dédiée pour la recherche et l'indexation.
Base de données	Hybris commerce prend en charge une variété de systèmes de bases de données. La réplication ou le clustering de base de données est recommandé pour le basculement et la haute disponibilité.
Serveur de fichiers	Hybris commerce utilise le système de fichiers pour stocker les images et autres éléments multimédias. Un NFS ou Storage Area Network (SAN) peut être utilisé pour stocker les ressources, évitant ainsi de devoir répliquer les fichiers sur tous les serveurs. Si les ressources statiques sont servies directement à partir des serveurs Web (par exemple, Apache), le système de fichiers partagé doit également être accessible aux serveurs Web.

Table. 9 : Description du rôle de chaque composant

La distribution de ces composants doit être pris sur l'évaluation de plusieurs critères (l'évolutivité, la haute disponibilité, la performance, la sécurité) et d'autres qualités du système pour garantir la configuration et la taille correctes.

À noter que les environnements utilisés pour le développement et les tests d'acceptation des utilisateurs ne nécessitent pas une grande taille. Ces environnements sont principalement utilisés pour les tests internes et ne doivent pas être chargés avec un grand nombre de demandes et de processus. Cela étant dit, ils ne devraient pas être sous-dimensionnés.

6.2. Environnement d'intégration

L'environnement d'intégration est utilisé par les développeurs afin de vérifier leurs changements du code source. Le déploiement à ses environnements s'effectue d'une manière automatique par Jenkins.

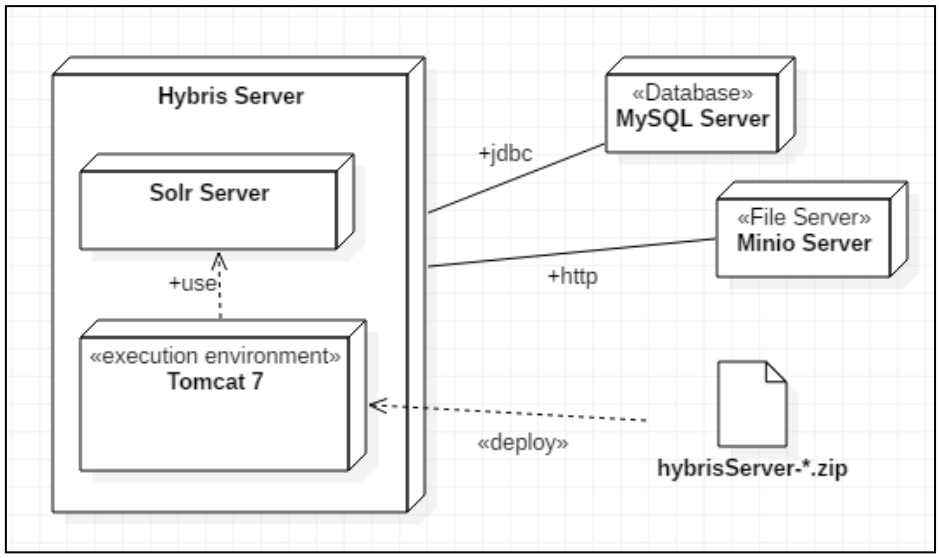


Figure. 18 : Digramme déploiement environnement d'intégration

6.3. Environnement de test

Cet environnement est utilisé par les testeurs pour vérifier les fonctionnalités et les nouveaux changements par des contrôles manuels ou automatisés. Les testeurs peuvent lancer des tâches de déploiement envers ces environnements.

L'architecture représentée dans le diagramme est proche de la production, mais pas similaire.

Si les tests passent dans ces environnements, l'application alors peut être dépoilée dans l'environnement de préparation.

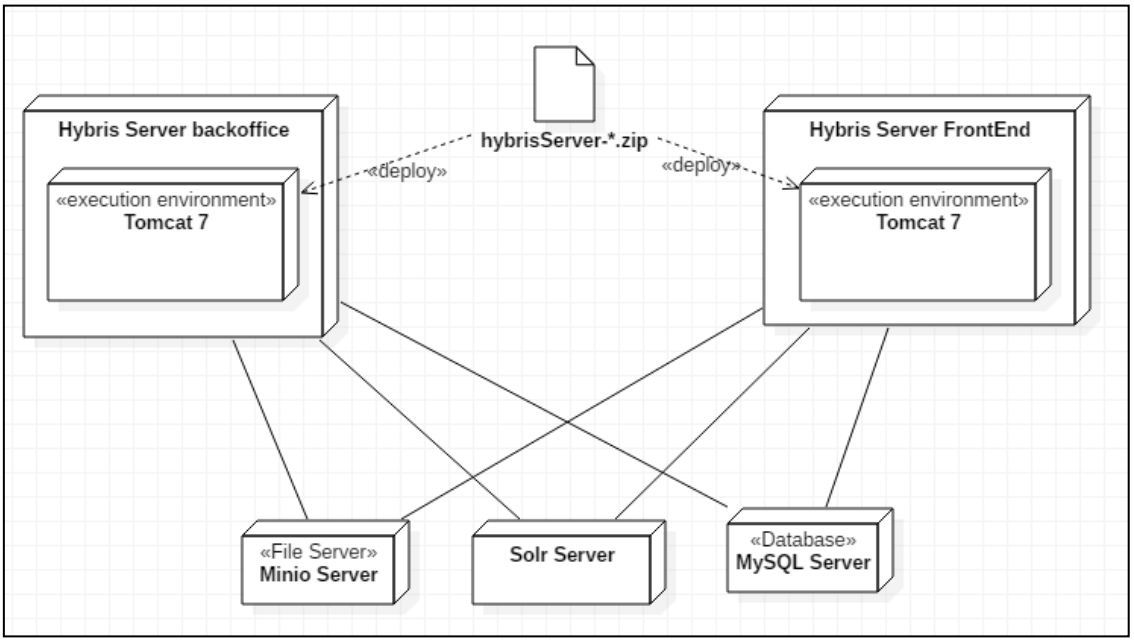


Figure. 19 : Digramme déploiement environnement de test

6.4. Environnement de préparation et production

Dans ce projet, les environnements production et préparation « pré production » sont similaires.

L'utilisation principale de l'environnement préparation est de tester tous les scripts et procédures d'installation, configuration et migration, avant qu'ils ne soient appliqués à l'environnement de production. Cela garantit que toutes les mises à niveau majeures et mineures de l'environnement de production seront effectuées de manière fiable et sans erreur, en un minimum de temps.

Alors que l'environnement de production est ou les serveurs dont va interagir le client et/ou seront les informations réelles des utilisateurs.

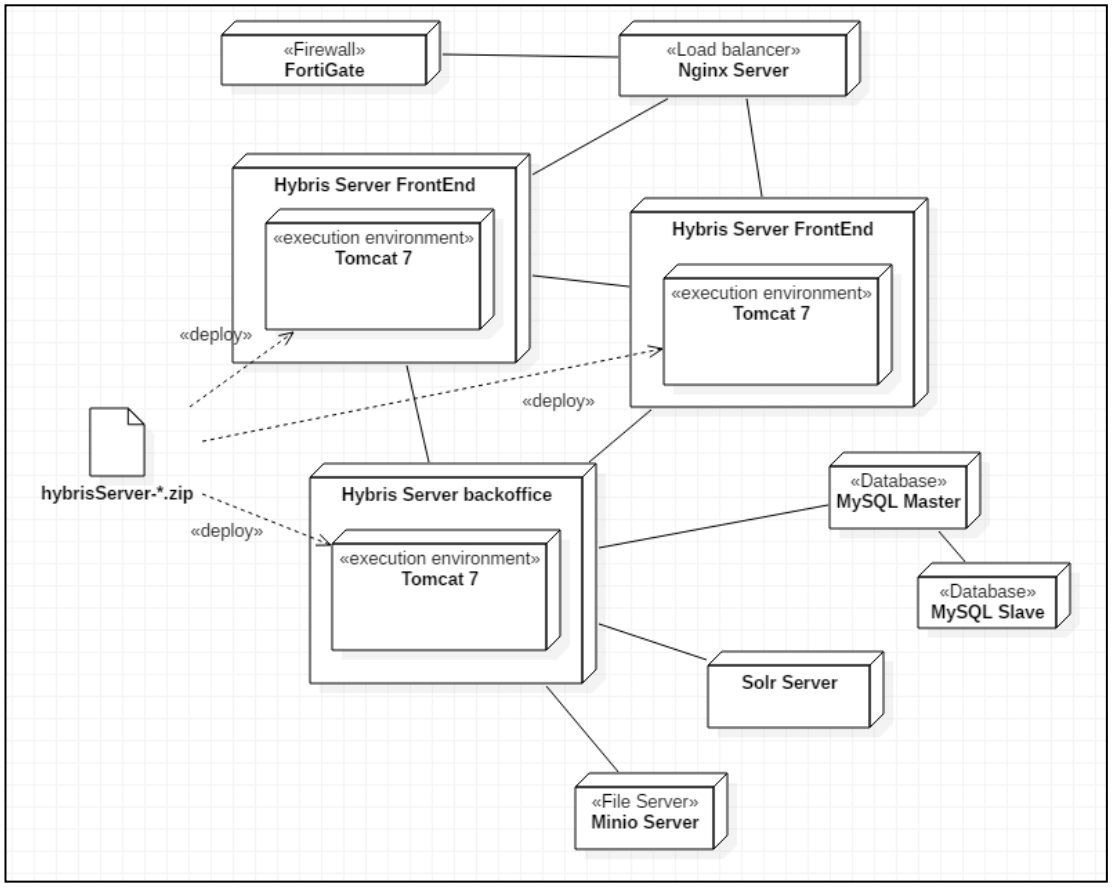


Figure. 20 : Digramme déploiement environnement de préparation et production

En fin, nous voyons la différence entre les environnements. Dans un premier lieu dans taille et nombre de nœuds dans le cluster Hybris. Dans un autre l'utilisation de réplication pour la base donnée MySQL.

De plus nous ajoutons que le cluster Hybris dans la production contient deux « fronts end » ce qui permet de réaliser un déploiement sans indisponibilité.

Conclusion

Ce chapitre nous a permis dans un premier lieu d'identifier les acteurs de notre système, ensuite nous avons spécifié les besoins fonctionnels et non fonctionnels de notre système avant de clôturer avec les architectures de déploiements des environnements du projet AMC et la plateforme de livraison continue.

Chapitre 4 : Réalisation

Introduction

Dans ce chapitre, nous mettons en place le pipeline de livraison continue amélioré. Ainsi que l'application des pratiques Pipeline as Code et Infrastructure as Code.

1. Mise en place du pipeline

La livraison continue permet aux organisations de fournir des logiciels à moindre risque. Le chemin vers la livraison continue commence par la modélisation du pipeline de livraison de logiciels utilisés au sein de l'organisation et se concentre ensuite sur l'automatisation de tout cela. La rétroaction directe précoce, activée par l'automatisation des pipelines, permet une livraison plus rapide des logiciels par rapport aux méthodes de livraison traditionnelles.

Jenkins est le couteau suisse dans la chaîne d'outils de livraison de logiciels. Le personnel des développeurs et des opérations (DevOps) ont des mentalités différentes et utilisent des outils différents pour accomplir leurs tâches respectives. Depuis Jenkins s'intègre à une grande variété d'outils, il sert de point d'intersection entre les équipes de développement et d'opérations.

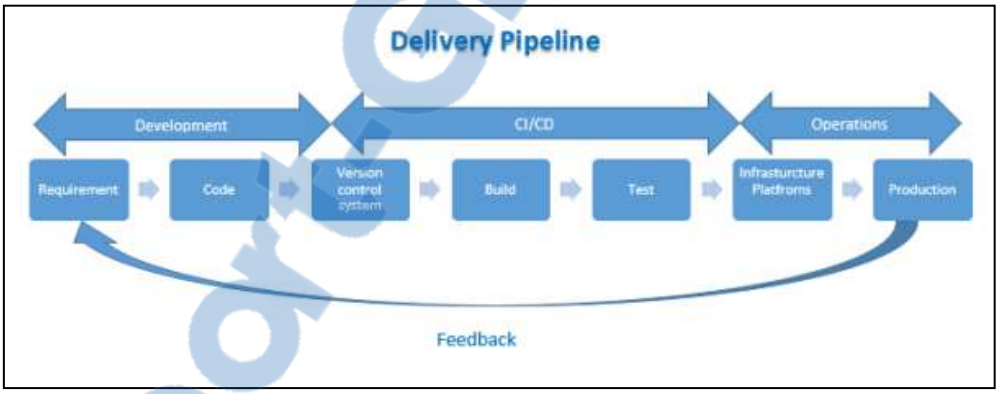


Figure. 21 : Processus de livraison continue

La livraison continue est un processus — plutôt que des outils — et nécessite un état d'esprit et une culture qui doivent percoler du haut vers le bas au sein d'une organisation. Une fois que l'organisation a adopté la philosophie, la partie suivante et la plus difficile consiste à cartographier le flux des logiciels au fur et à mesure qu'ils passent du développement à la production [14].

Dans notre cas, nous avons réalisé une étude comparative des outils DevOps pour choisir les meilleurs et les plus adaptés aux besoins de SQLI. Ce qui nous a donné une architecture globale.

Dans les parties qui suivent, nous précédons à la mise en place d'outils.

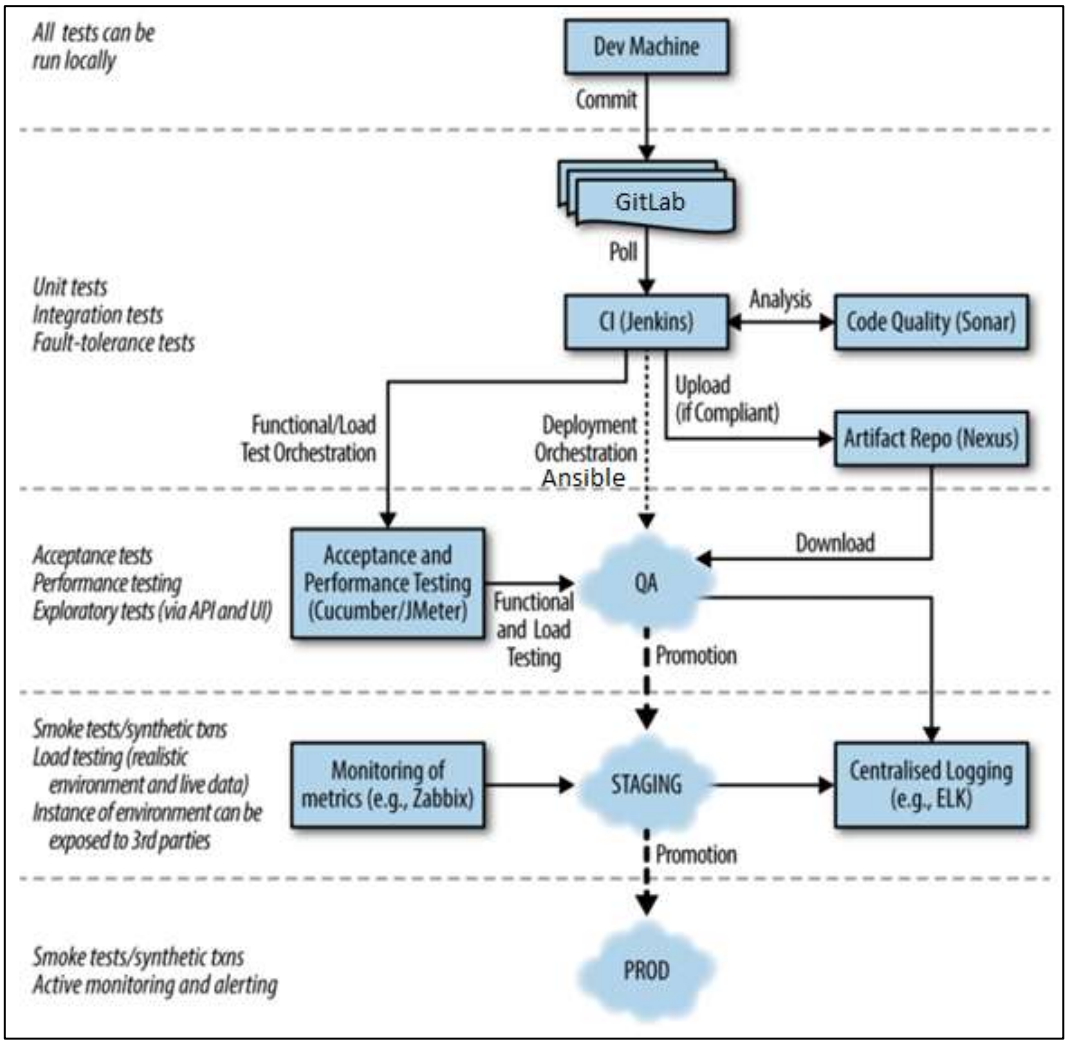


Figure. 22 : Architecture globale des outils de livraison continue [15]

La figure illustre un pipeline de livraison continue typique pour une application basée sur Java. La première étape du processus de CD est l'intégration continue. Le code créé sur l'ordinateur portable d'un développeur est continuellement intégré dans un référentiel de contrôle de version partagé et commence automatiquement son parcours dans tout le pipeline.

L'objectif principal du pipeline de construction est de prouver que les changements sont prêts pour la production. Un code de modification de la configuration peut échouer à n'importe quel stade du pipeline, et cette modification sera par conséquent rejetée et non marquée comme prête pour le déploiement en production.

Initialement, l'application logicielle à laquelle un changement de code est appliqué est construite et testée isolément, et une certaine forme d'analyse de la qualité du code peut (devrait) également être appliquée, peut-être en utilisant un outil tel que SonarQube. Le code qui passe avec succès les tests unitaires et composants initiaux et les métriques de qualité de code se déplacent vers la droite dans le pipeline et s'exerce dans un contexte intégré plus

large. Finalement, le code qui a été entièrement validé émerge du pipeline et est marqué comme prêt pour le déploiement en production. Certaines organisations déploient automatiquement des applications qui ont réussi à naviguer dans le pipeline de génération et à passer tous les contrôles de qualité, ce qui est appelé « livraison continue ».

1.1. GitLab

Après installation de GitLab, nous configurons LDAP pour avoir l'authentification centralisée et STMP pour recevoir les notifications depuis la page d'administration.

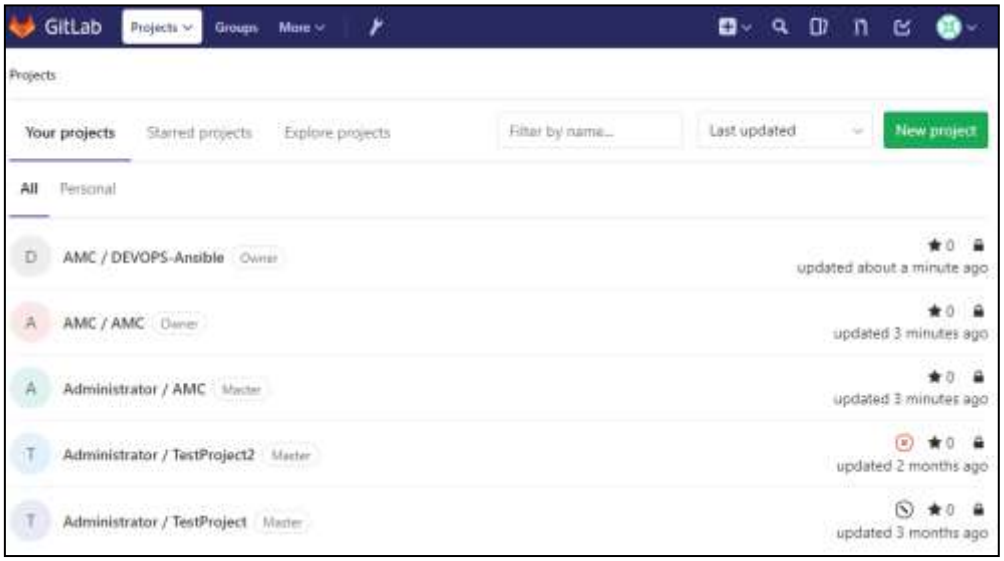


Figure. 23 : Page d'Accueil GitLab

Ainsi pour crée un dépôt de code source pour le projet AMC.

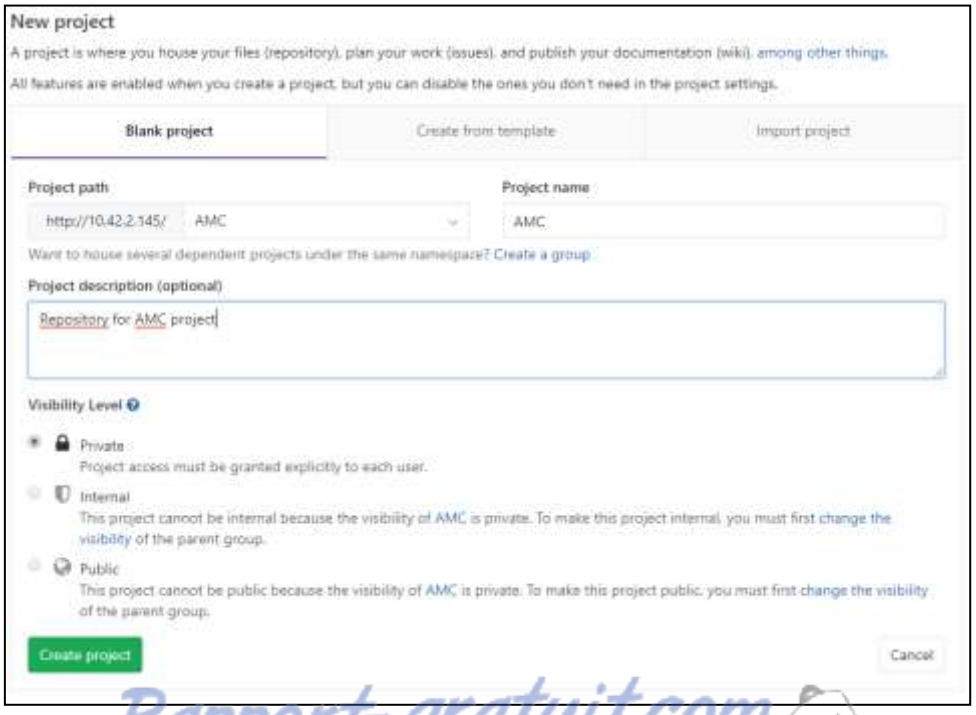


Figure. 24 : Création de repository AMC

1.2. Jenkins

Après installation de Jenkins, nous configurons LDAP et STMP, ainsi que les outils des constructions comme Maven et Ant. En fin, nous ajoutons les esclaves Jenkins.

Afin de recevoir les notifications en courrier électronique depuis Jenkins, nous devons configurer les paramétrées de STMP, dans la vue gestion Jenkins sous la section notification e-mail.

Figure. 25 : Configuration de STMP

Nous configurons l'outil de construction Maven dans la vue globale de configuration des outils.

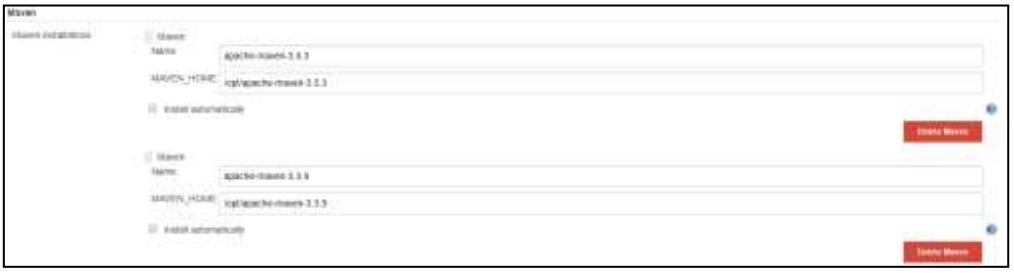


Figure. 26 : Configuration Outil construction Maven

Nous terminons par l'ajout des Slaves que nous avons déjà cité dans la conception, comme suite.

Figure. 27 : Configuration esclave Jenkins

1.3. SonarQube

Après installation de SonarQube, nous configurons LDAP et STMP depuis la page d'administration. Alors que pour la configuration LDAP, nous ajoutons au fichier *SONARQUBE_HOME/conf/sonar.properties*.

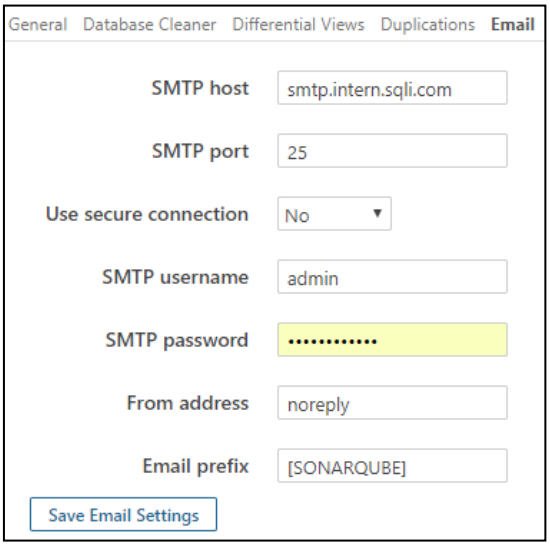


Figure. 28 : Configuration STMP SonarQube

```
# LDAP configuration
# General Configuration
sonar.security.realm=LDAP
ldap.url=ldap://ldap.internsqli.com
ldap.bindDn=admin
ldap.bindPassword=
```

Figure. 29 : Configuration LDAP SonarQube

De suite nous créons une « Quality Gate » pour le projet AMC avec des conditions sur la qualité de code source et le courage des tests unitaires. Ces conditions doivent être respectées ou le pipeline échouera. Dans l'onglet « Quality Gate », nous cliquons sur ajouter nouveau.

Figure. 30 : Quality Gate pour projet AMC

1.4. Nexus

Après l'installation de Nexus, nous configurons LDAP et STMP depuis la page d'administration.

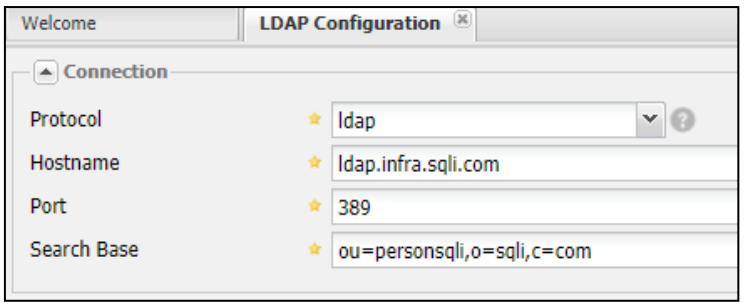


Figure. 31 : Configuration LDAP Nexus

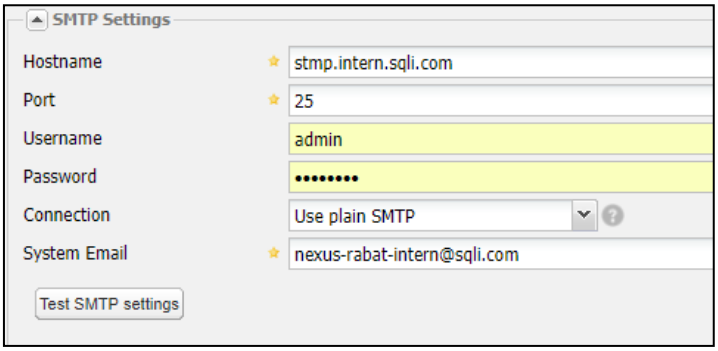


Figure. 32 : Configuration STMP Nexus

Et à titre d'exemple, nous créons un dépôt d'artefacts Nexus de type Maven pour le projet AMC. Ce dépôt va être utilisé par la suite pour publier les artéfacts d'Hybris. Dans l'onglet dépôts, nous cliquons sur ajouter.

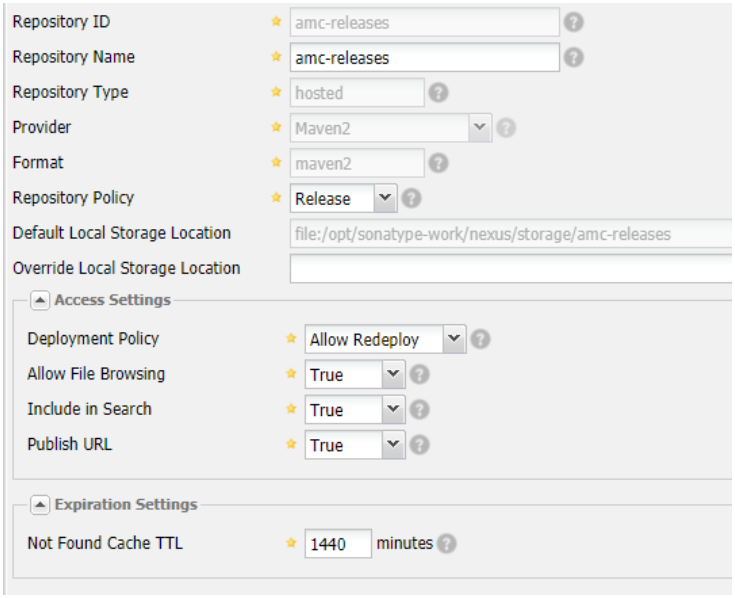


Figure. 33 : Création de référence Nexus pour AMC

1.5. Ansible

Afin de mettre en place, Ansible nous l'installons sur la machine de Jenkins. Ou l'avoir installé sur machine esclave de Jenkins. Nous avons choisi la première méthode pour sa facilité.

1.5.1. Installation et configuration

Pour installer Ansible, nous trouvons plusieurs méthodes. La méthode la plus flexible c'est d'utiliser le système de gestion de package PIP, au lieu d'utiliser le gestionnaire de package du système d'exploitation. Pour installer une version donnée dans la machine Jenkins, nous lançons la commande : *user@jenkins# sudo pip2 install ansible==2.1*

après nous ajoutons le plug-in d'Ansible dans Jenkins, pour se faire, depuis le menu gestion de Jenkins, nous choisissons l'option gestion de plug-ins. Dans l'onglet disponible, nous cherchons Ansible. Ensuite, nous avons coché la case adjacente au plug-in et cliqué sur Installer sans redémarrer.

Figure. 34 : Installation Jenkins Ansible plug-in

Après l'installation du plug-in, pour le configurer nous allons à la gestion des outils dans l'interface web (gérer Jenkins> Vue Configuration globale de l'outil). Dans la section Ansible, nous ajoutons notre installation Ansible. Nous définissons le nom et le chemin d'accès au répertoire exécutable Ansible.

Figure. 35 : Configuration outil Ansible dans Jenkins

1.5.2. Configuration des nœuds

Pour exécuter les Playbooks Ansible à partir de Jenkins, nous devons configurer un accès SSH sans mot de passe entre le serveur Jenkins et les machines nœuds des environnements.

Parce que nous avons besoin d'une forme de connexion automatique du serveur Jenkins aux machines nœuds. Nous devons également configurer le lancement de la commande « sudo » sans mot de passe pour l'utilisateur distant.

- SSH login sans mot de passe : Pour chaque nœud, nous utilisons *ssh-copy-id* qui ajoute la clé publique à l'utilisateur distant *~/.ssh /authorized_keys*.
  jenkins@server:~ $ ssh-copy-id — i ~/. ssh/id_rsa.pub user@hostname
- La commande Sudo sans mot de passe : Pour activer *sudo* sans mot de passe, nous devons éditer le fichier/*etc/sudoers*, sur chaque nœud. En ajoutant la ligne suivante :
  $ echo « user ALL=(ALL : ALL) NOPASSWD:ALL » | sudo tee -a /etc/sudoers

1.5.3. Configuration tâche Ansible dans Jenkins

Cette configuration d'une tâche normale qui exécute Ansible peut être utilisée dans les cas simples sans utiliser la ligne de commande. La configuration de la tâche est comme suit. Dans la vue de configuration de la tâche Jenkins, nous ajoutons le référentiel git de Playbook Ansible.

Figure. 36 : Ajout du dépôt git tâche Ansible

Et dans l'étape de construction du travail Jenkins, nous sélectionnons « Invoke Ansible Playbook » et nous spécifions dans les entrées du nom d'installation Ansible, le chemin du Playbook, le chemin du fichier d'inventaire.

Figure. 37 : Configuration exécution Ansible depuis tâche Ansible

Après l'exécution d'une tâche, nous avons un journal d'exaction. Ce journal est la sortie de lacement du Playbook sur les machines d'inventaire.

```
[Update local.properties file] $ /usr/bin/ansible-playbook plays/deploy-properties.yml
file/vault6518095865564174857.password"

PLAY [webservers] ***************************************************************

TASK [setup] ********************************************************************
ok: [10.42.3.120]
ok: [10.42.2.198]

TASK [hybris-deploy-properties : Ensure Hypris config directory exist] *********
ok: [10.42.3.120]
ok: [10.42.2.198]

TASK [hybris-deploy-properties : Update local.properties] **********************
ok: [10.42.2.198]
ok: [10.42.3.120]

TASK [hybris-deploy-properties : Ensure Update proprieties cron job exist] *****
changed: [10.42.2.198]
changed: [10.42.3.120]

PLAY RECAP **********************************************************************
10.42.2.198                : ok=4    changed=1    unreachable=0    failed=0
10.42.3.120                : ok=4    changed=1    unreachable=0    failed=0

Finished: SUCCESS
```

Figure. 38 : Journal d'exécution d'une tâche Ansible

Nous expliquons par plus de détails les concepts de base d'Ansible dans la partie gestion de configuration dans ce chapitre.

1.5.4. Ansible Vault pour les informations sensitives

Le « Vault » est une fonctionnalité d'Ansible qui nous permet de conserver des données sensibles telles que les mots de passe ou les clés dans des fichiers cryptés, plutôt que de les laisser en clair dans les Playbooks ou les rôles. Ces fichiers cryptés peuvent ensuite être distribués ou placés dans le contrôle de source.

Pour exécuter un Playbook Ansible avec des fichiers de données cryptés par Vault, nous devons fournir le mot de passe Vault. Pour notre cas, ce mot de passe est stocké dans vault_pass.txt. Alors pour chiffrer ou déchiffrer un fichier de Vault :

 # ansible-vault {encrypt|decrypt} --vault-password-file={vault pass file} {vault file}

Pour utiliser cette fonctionnalité avec Jenkins Ansible Plugin, nous devons fournir de même le fichier vault_pass.txt.

Figure. 39 : Ajout d'un fichier secret dans Jenkins

2. Tâche pipeline dans Jenkins

Le modèle d'interaction par défaut avec Jenkins, historiquement, a été très axé sur l'interface utilisateur Web, obligeant les utilisateurs à créer manuellement des tâches, puis à remplir manuellement les détails via un navigateur Web. Cela nécessite des efforts supplémentaires pour créer et gérer des tâches afin de tester et de générer plusieurs projets. Il conserve également la configuration d'une tâche à construire, tester et déployer séparément du code en cours de construction, test et déploiement. Cela empêche les utilisateurs d'appliquer leurs meilleures pratiques de livraison continue existantes aux configurations de travail elles-mêmes.

2.1. Pipeline as Code

Pipeline as Code décrit un ensemble de fonctionnalités qui permettent aux utilisateurs de Jenkins de définir des processus de travail en pipeline avec code, stocké et versionné dans un référentiel source. Ces fonctionnalités permettent à Jenkins de découvrir, gérer et exécuter des tâches pour plusieurs référentiels et branches sources, éliminant ainsi la nécessité de créer et de gérer manuellement les travaux [12].

Pour utiliser Pipeline comme code, les projets doivent contenir un fichier nommé Jenkinsfile dans la racine du référentiel, qui contient un « script de pipeline ».

2.1. Jenkinsfile

La présence du fichier Jenkinsfile à la racine d'un référentiel permet à Jenkins de gérer et d'exécuter automatiquement les tâches en fonction des branches du référentiel.

Le fichier Jenkins doit contenir un « script de pipeline », en spécifiant les étapes d'exécution du travail. Le script a toute la puissance de Pipeline disponible, de quelque chose d'aussi simple que l'appel d'un constructeur Maven, à une série d'étapes interdépendantes, qui ont coordonné une exécution parallèle avec des phases de déploiement et de validation.

2.2. Jenkinsfile Projet AMC

Notre Jenkinsfile du projet AMC contiendra plusieurs étapes chaque étape effectue des tâches spécifiques. Ces tâches sont :

- Compilation et construction
- Test unitaire
- Analyse qualimétrie et publication rapport
- Empaquetage des artéfacts
- Publication des artéfacts sur Nexus
- Déploiement sur environnement

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'cd hybris && mvn -B compile'
            }
        }
        stage('Unit Tests') {
            steps {
                sh 'cd hybris && mvn -B  -DskipCompile process-classes'
            }
        }
        stage('Sonar') {
            steps {
                sh 'cd hybris && mvn -B -DskipCompile -Dsonar=true process-classes'
            }
        }
        stage('Package') {
            steps {
                sh 'cd hybris && mvn -B -DskipCompile package'
            }
        }
        stage('Publish') {
            steps {
                sh 'cd hybris && mvn -U -B -DskipProduction deploy'
            }
        }
        stage('Deploy') {
            steps {
                sh 'cd hybris && mvn -U -B -DskipProduction site-deploy'
            }
        }
    }
}
```

Figure. 40 : Jenkinsfile projet AMC

Ce fichier Jenkinsfile est exécuté à chaque fois qu'il y a un changement au code source. À noter que l'exemple ci-dessus ne contient pas tous les détails. Dans la figure ci-dessous, on trouve un exemple d'exécution.

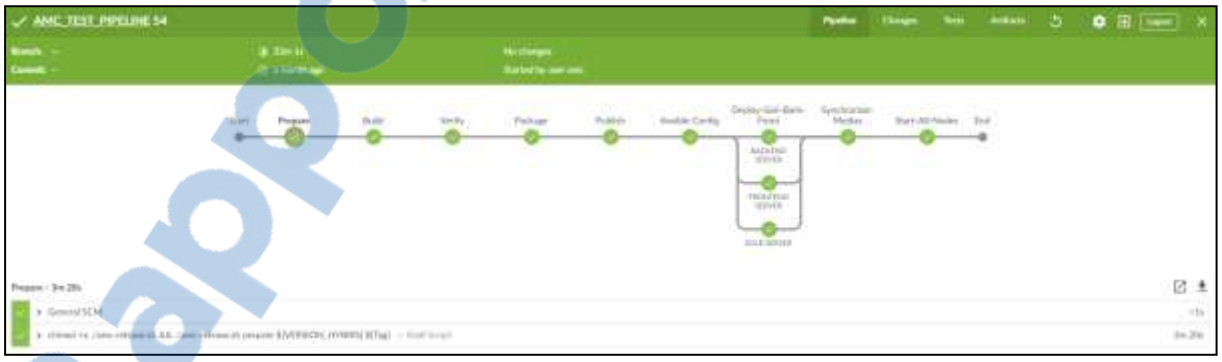


Figure. 41 : Tâche pipeline projet AMC

Après le succès des étapes constructions, test unitaire, empaquetage et publication des artéfacts sur Nexus. Nous passons à l'étape de déploiement des artéfacts. Ces artéfacts sont de la plateforme e-commerce Hybris, projet AMC. Pour les déployer, nous utilisons un « playbook » Ansible, dont nous allons donner le détail dans la prochaine partie.

Dans la figure suivante, nous voyons la publication des artéfacts dans le dépôt d'objet binaire de type Maven, nommé « amc-releases ».

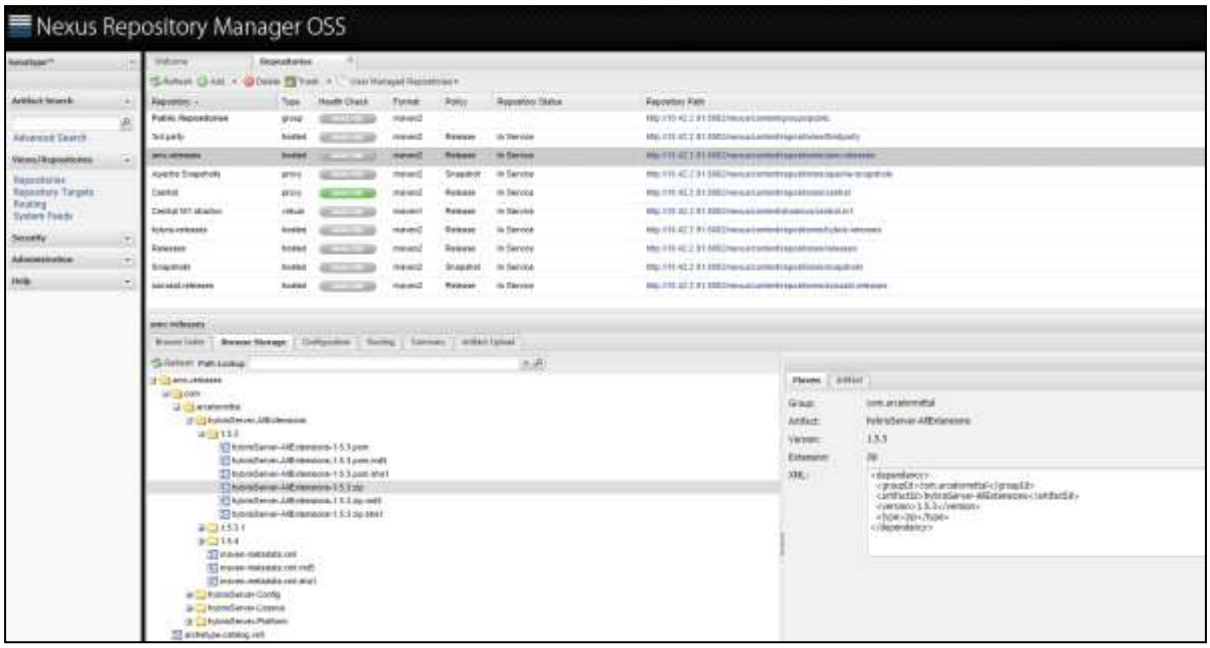


Figure. 42 : page d'accueil Nexus et liste d'artéfacts

De même cette figure ci-dessous, nous montrons les critères de qualité de code pour le projet AMC, ce rapport est actualisé à chaque analyse. À noter que si le projet dépasse la borne de qualité « Quality Gate » la tâche pipeline échouera.

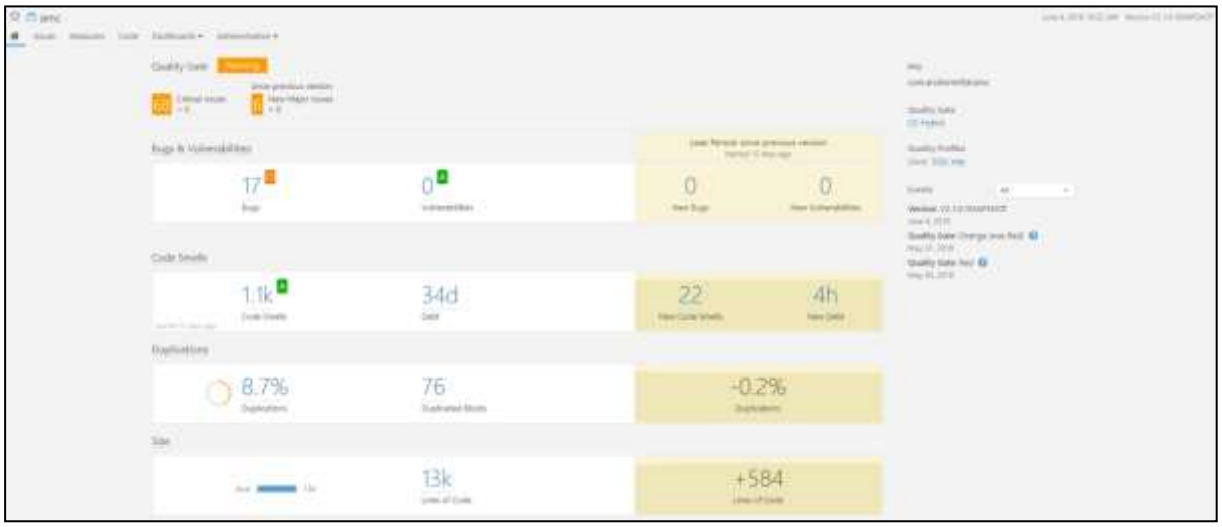


Figure. 43 : Rapport d'analyse qualimétrie

D'après ce que nous venons de montrer, ce pipeline de livraison n'est pas encore mature en terme de tests. Pas de tests d'intégration et de régression après les tests unitaires. Aussi, les tests fonctionnels ne sont pas automatisés et se font manuellement par les testeurs. De plus les tests de performances et d'acceptation ne sont pas encore implémentés. Dans l'état actuel (Début du projet) peut être acceptable, mais les tests doivent être mise en place dans le futur.

Autre point à signaler, à part le déploiement en continu dans les environnements d'intégration, le déploiement dans les autres environnements est manuelle est exécuté par les

utilisateurs (Testeur, Chef projet, Développeur). Alors que dans les meilleures pratiques de livraison continue le déploiement doit être automatique jusqu'au les environnements de préparation « pré production » à chaque changement.

3. Gestion de configuration et Automatisation déploiement avec Ansible

Dans les prochains paragraphes, nous allons découvrir l'architecture d'Ansible et son fonctionnement. Puis nous montons la gestion de configuration des environnements et le déploiement du projet AMC avec Ansible.

3.1. Architecture Ansible

Ansible est un moteur d'automatisation simple qui automatise le Cloud, le provisionnement, la gestion de la configuration, le déploiement d'applications, l'orchestration de services ainsi que beaucoup d'autres fonctionnalités [14].

Dans cette section, nous allons vous donner un aperçu de la façon dont Ansible fonctionne.

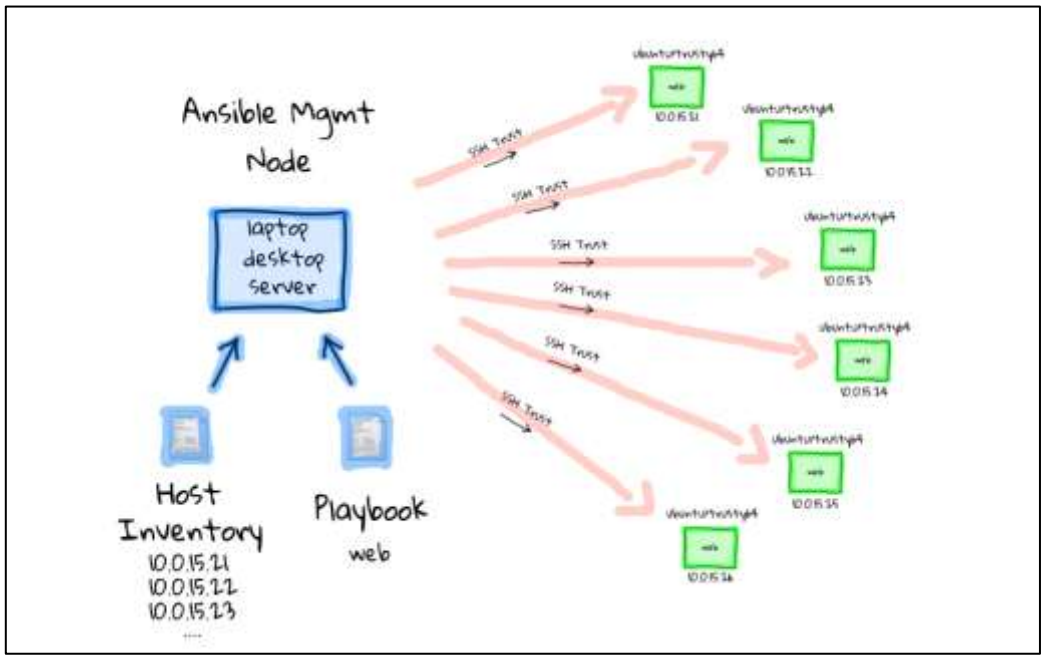


Figure. 44 : Architecture Ansible

3.1.1 Module

Ansible fonctionne en se connectant aux nœuds et en poussant de petits programmes, appelés « Ansible Modules ». Ces programmes sont écrits pour être des modèles de ressources de l'état souhaité du système. Ansible exécute ensuite ces modules (sur SSH par défaut) et les supprime une fois terminé. La bibliothèque de modules peut résider sur n'importe quel ordinateur et aucun serveur, démon ou base de données n'est requis.

3.1.2 Inventaire

L'inventaire fournit la liste des hôtes (serveurs managés) sur lesquels Ansible exécutera les tâches. Il peut également être utilisé pour regrouper les hôtes et configurer les variables pour les hôtes et les groupes.

```
---
[webservers]
front.amc.sqli.com
back.amc.sqli.com

[dbservers]
db.amc.sqli.com
```

Figure. 45 : Exemple fichier Ansible Inventory

3.1.3. Playbooks

Le Playbook est le langage de configuration d'Ansible utilisé pour effectuer l'ensemble des tâches et d'étapes de configuration ou pour faire appliquer une politique sur les nœuds distants. Les modules Ansible sont utilisés dans le Playbook pour exécuter une opération.

Le Playbook est écrit en YAML qui est un langage simple, lisible par l'homme et développé dans un anglais de base comme la langue du texte. Les Playbooks sont plus susceptibles d'être conservés dans le système de contrôle de version et utilisés pour assurer les configurations des systèmes distants.

Les Playbooks sont utilisés de la simple configuration et la gestion des nœuds distants jusqu'au déploiement avancé impliquant des mises à jour, le suivi des serveurs, la répartition de charge, etc.

```
---
-  name: install and start Apache webserver
   hosts: webservers
   user: root

   tasks:
      name: install httpd
      yum: name=httpd state=present

      name: start httpd
      service: name=httpd state=running
```

Figure. 46 : Exemple Playbook pour installation Apache

3.2. Automatisation Déploiement SAP Hybris e-commerce plateforme

Pour déployer les artéfacts d'Hybris e-commerce plateforme, nous suivons plusieurs étapes :

- Arrêter le serveur d'application Hybris

- Crée un backup de la version installée
- Téléchargement des artéfacts HybrisServer— *. zip de la nouvelle version depuis Nexus
- Décompression des artéfacts dans le répertoire/*opt/hybris/*(doit être vide)
- Lancer le serveur d'application Hybris

Nous avons donné les étapes à suivre dans le cas d'un déploiement dans un seul serveur. Dans le cas d'un cluster Hybris, la procédure se lance sur un nœud à la fois.

Pour réaliser cette procédure avec Ansible, nous avons créé un Playbook qui s'exécute sur un groupe serveur web dans un inventaire.

3.2.1. Inventaire

Nous donnons un exemple de l'inventaire de l'environnement de test. Cet environnement contient deux serveurs Hybris Frontend et Backoffice. Alors ces deux derniers qui seront affectés par le déploiement et sont regroupés par Web servers.

hosts 112 Bytes

```
1  [admin]
2  10.42.3.120
3
4  [front]
5  10.42.2.198
6
7  [webservers:children]
8  admin
9  front
10
11 [db]
12 10.42.3.19
13
14 [solr]
15 10.42.3.19
```

Figure. 47 : Inventaire environnement test

3.2.2. Playbook déploiement Hybris

Dans ce qui suit nous donnons une version du Playbook de déploiement Hybris qui appliquer les étapes déjà citées sur les nœuds web servers.

deploy.yml 189 Bytes

```yaml
---

- hosts: webservers
  become: yes
  remote_user: hybris
  become_method: sudo
  serial: 1 # Rolling Update, one host at a signle time

  roles:
    - hybris-deploy
```

Figure. 48 : Playbook déploiement Hybris

Le Playbook fait appelle un rôle Ansible, un rôle est un ensemble des tâches et opérations organisé selon une structure standardisée pour offrir plus de modularité et réutilisabilité. Ce rôle nommé *hybris-deploy* contient plusieurs tâches [cf Figure. 49].

L'option serial est égale à un, ce qui impose à Ansible de lancer les opérations sur un nœud à la fois.

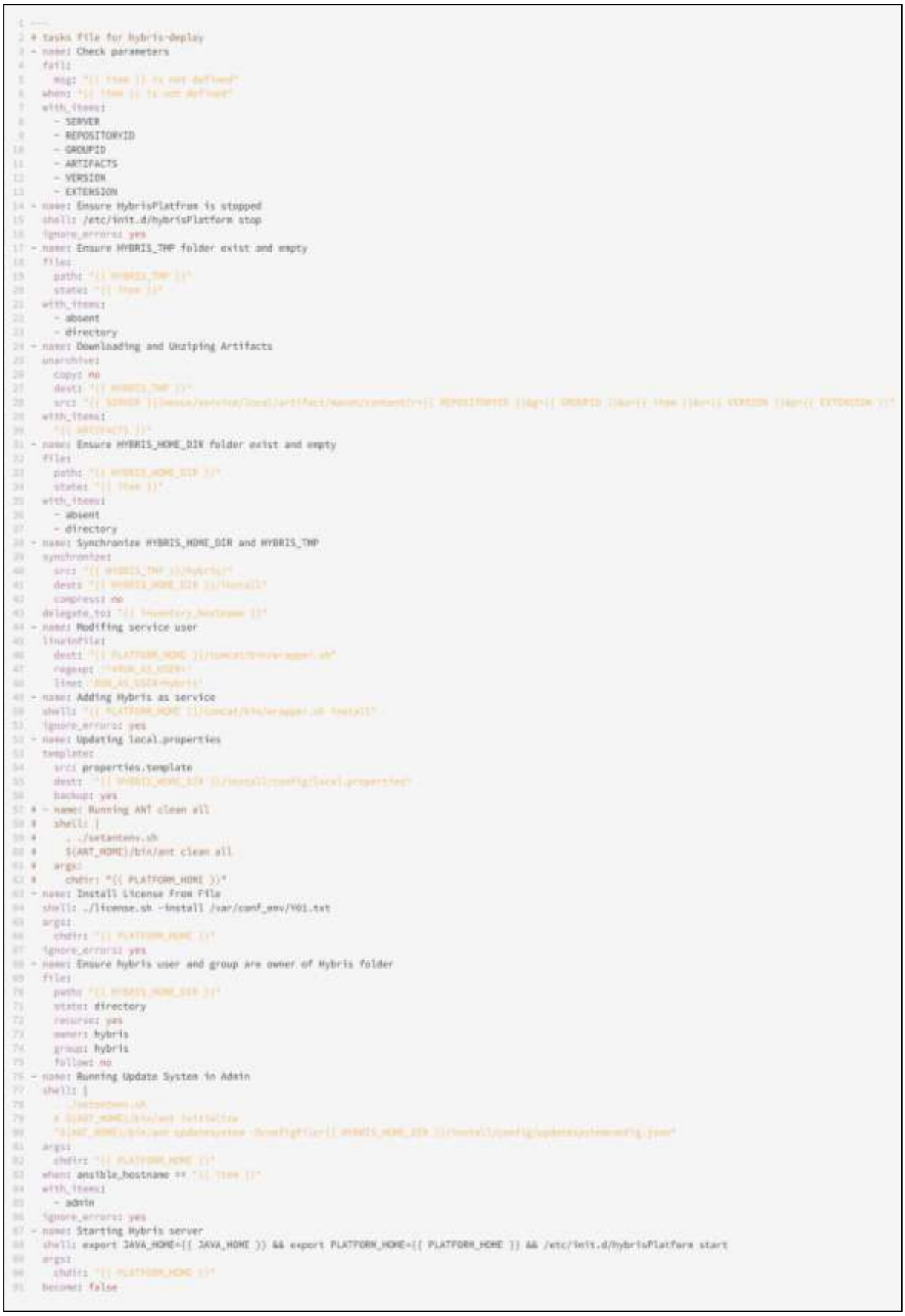
3.3.2. Exécution

Pour exécuter ou lancer le Playbook sur un serveur ou ensemble de serveur Hybris, nous devons renseigner les paramètres suivants :

- SERVER : nom d'hôte serveur Nexus
- REPOSITORYID : l'identifiant de dépôt Nexus
- GROUPID : Le nome de projet
- ARTIFACTS : Une liste des artéfacts
- VERSION : La version des artéfacts

Cela rend ce Playbook applicable à tous les projets qui se base sur la plateforme Hybris, et non pas seulement le projet AMC. La commande suivante est un exemple d'exécution du Playbook :

```
ansible-playbook -l front -i hosts plays/deploy.yml
'{"SERVER": "'${SERVER}'","REPOSITORYID":"amcreleases", "GROUPID":"com.arcelormittal",
"ARTIFACTS": ["hybrisServer-Platform","hybrisServer-AllExtensions","hybrisServer-Config"],
"VERSION": "'${HYBRIS_VERSION}'", "EXTENSION": "zip"}'
--vault-password-file=../vault_pass.txt
```



```
---
# tasks file for hybris-deploy
- name: Check parameters
  fail:
    msg: "{{ item }} is not defined"
  when: "{{ item }} is not defined"
  with_items:
    - SERVER
    - REPOSITORYID
    - GROUPID
    - ARTIFACTS
    - VERSION
    - EXTENSION
- name: Ensure HybrisPlatfrom is stopped
  shell: /etc/init.d/hybrisPlatform stop
  ignore_errors: yes
- name: Ensure HYBRIS_TMP folder exist and empty
  file:
    path: "{{ HYBRIS_TMP }}"
    state: "{{ item }}"
  with_items:
    - absent
    - directory
- name: Downloading and Unziping Artifacts
  unarchive:
    copy: no
    dest: "{{ HYBRIS_TMP }}"
    src: "{{ SERVER }}/nexus/service/local/artifact/maven/content?r={{ REPOSITORYID }}&g={{ GROUPID }}&a={{ item }}&v={{ VERSION }}&p={{ EXTENSION }}"
  with_items:
    - "{{ ARTIFACTS }}"
- name: Ensure HYBRIS_HOME_DIR folder exist and empty
  file:
    path: "{{ HYBRIS_HOME_DIR }}"
    state: "{{ item }}"
  with_items:
    - absent
    - directory
- name: Synchronize HYBRIS_HOME_DIR and HYBRIS_TMP
  synchronize:
    src: "{{ HYBRIS_TMP }}/hybris/"
    dest: "{{ HYBRIS_HOME_DIR }}/hybris"
    compress: no
  delegate_to: "{{ inventory_hostname }}"
- name: Modifing service user
  lineinfile:
    dest: "{{ PLATFORM_HOME }}/tomcat/bin/wrapper.sh"
    regexp: '^RUN_AS_USER='
    line: 'RUN_AS_USER=hybris'
- name: Adding Hybris as service
  shell: "{{ PLATFORM_HOME }}/tomcat/bin/wrapper.sh install"
  ignore_errors: yes
- name: Updating local.properties
  template:
    src: properties.template
    dest: "{{ HYBRIS_HOME_DIR }}/install/config/local.properties"
    backup: yes
# - name: Running ANT clean all
#   shell: |
#     . ../setantenv.sh
#     ${ANT_HOME}/bin/ant clean all
#   args:
#     chdir: "{{ PLATFORM_HOME }}"
- name: Install License From File
  shell: ./license.sh -install /var/conf_env/Y01.txt
  args:
    chdir: "{{ PLATFORM_HOME }}"
  ignore_errors: yes
- name: Ensure Hybris user and group are owner of Hybris folder
  file:
    path: "{{ HYBRIS_HOME_DIR }}"
    state: directory
    recurse: yes
    owner: hybris
    group: hybris
    follow: no
- name: Running Update System in Admin
  shell: |
      . ./setantenv.sh
    # ${ANT_HOME}/bin/ant initialize
    "${ANT_HOME}/bin/ant updatesystem -DconfigFile={{ HYBRIS_HOME_DIR }}/install/config/updatesystemconfig.json"
  args:
    chdir: "{{ PLATFORM_HOME }}"
  when: ansible_hostname == "{{ item }}"
  with_items:
    - admin
  ignore_errors: yes
- name: Starting Hybris server
  shell: export JAVA_HOME={{ JAVA_HOME }} && export PLATFORM_HOME={{ PLATFORM_HOME }} && /etc/init.d/hybrisPlatform start
  args:
    chdir: "{{ PLATFORM_HOME }}"
  become: false
```

Figure. 49 : Tâches du rôle Ansible hybris-deploy

3.3. Gestion de configuration des environnements

L'automatisation joue un rôle essentiel pour nous permettre de diffuser le logiciel de manière répétée et fiable. Un objectif clé consiste à prendre les processus manuels répétitifs, tels que la génération, le déploiement, les tests de régression et le provisionnement d'infrastructure, et à les automatiser.

Pour ce faire, nous devons contrôler tous les éléments requis pour exécuter ces processus, notamment le code source, les scripts de test et de déploiement, l'infrastructure et les informations de configuration des applications, ainsi que les nombreuses bibliothèques et packages dont nous dépendons. Nous voulons également simplifier la recherche de l'état actuel et historique de nos environnements.

Dans notre cas nous utilisons Ansible pour gérer la configuration des environnements du projet AMC. Chaque machine dans un environnement à des besoins et des dépendances spécifiques que nous devons gère d'une manière ou d'autres. Avec les outils de gestion de configuration, nous décrivons l'état d'une ressource et l'outil s'empare de garder la machine dans cet état.

Nous prenons l'exemple du serveur Hybris de l'environnement de Test, à fin que Hybris peut se déployer sur ce serveur, le serveur doit respecter plusieurs points :

- Application Oracle Java installé
- Utilisateur *hybris* existe
- Application ImageMagick installé
- Les répertoires initiaux de la plateforme Hybris existent

Pour répondre à ce besoin de pré-configuration de machine, nous avons créé un Playbook qui s'exécute sur le groupe web servers de l'inventaire environnement de Test [cf. Partie 3.2.1].

3.3.1. Playbook configuration serveur Hybris

De même que le Playbook de déploiement, ce Playbook fait appelle à un rôle nommé *Hybris-site*.

site.yml 234 Bytes

```
---
# This playbook is to preconfigure a hybris hoste for deployment

- hosts: webservers
  remote_user: nwna
  become: true
  become_method: sudo
  become_user: root
  strategy: free

  roles:
    - { role: hybris-site, become: yes }
```

Figure. 50 : Playbook configuration serveur Hybris

Nous voyons que ces tâches utilisent des modules afin de rendre le serveur dans l'état désiré.

```
---
- name: Ensure prerequisites are installed
  package:
    name: "{{ item }}"
    state: present
  with_items:
    - sudo
    - zip
    - unzip
    - tar
    - rsync
- name: Ensure hybris user is created
  user:
    name: hybris
    createhome: yes
    home: /home/hybris
    system: yes
    password: $1$hyhrhis5$rZFCKmJj9maLitEnLgqQf1
    update_password: always
- name: Ensure members of group hybris to excute any command
  copy:
    dest: /etc/sudoers.d
    src: hybris
- include: install-oracle-java.yml
- include: install-imagemagick.yml
- name: Ensure initial directories exist
  file:
    path: "{{ item }}"
    state: directory
    owner: hybris
    group: hybris
  with_items:
    - /opt/amc
    - /opt/ImageMagick/
    - /opt/wkhtmltox
    - /opt/import
    - /opt/pdfgeneration/
    - /var/env_conf/
    - "{{ HYBRIS_TMP }}"
    - "{{ HYBRIS_HOME_DIR }}"
    - "{{ HYBRIS_INSTALL_HOME }}"
    - "{{ HYBRIS_CONFIG_DIR }}"
    - "{{ TOMCAT_CONFIG_DIR }}"
```

Figure. 51 : Tâche du rôle Ansible hybris-site

3.3.2. Exécution

Pour lancer ce Playbook, nous utilisons tous simplement la commande suivante :

ansible-playbook -i hosts plays/site.yml --vault-password-file=.. /vault_pass.txt

Nous avons vu que la configuration du serveur Hybris, alors que toutes les autres machines sont configurées (MySQL, Solr, etc.) avec Ansible en utilisant des rôles déjà développés. Dans ce cadre, nous citons le site Ansible Galaxy qui est une bibliothèque des rôles qui peut être utilisé ou modifier dans quelque cas pour correspondre notre propre exigence.

Ces Playbooks sont versionnés et ajoutés au code source, et décrivant l'état de notre infrastructure physique. Ce qui entre dans la pratique d'infrastructure en tant que code.

Conclusion

Ce dernier chapitre porte sur la réalisation des objectifs de notre travail, ainsi après la mise en place du pipeline. Nous montrons l'agrégation des tâches Jenkins dans une seule tâche de type Pipeline. Et nous terminons par la gestion de configuration des environnements avec Ansible.

Conclusion Générale

Dans le cadre de ce projet de fin d'études, nous étions amenés à améliorer un pipeline de livraison continue en intégrant l'outil de gestion de configuration et d'automatisation Ansible.

Après une étude comparative des outils DevOps, nous avons choisi les outils les plus adaptés aux besoins de SQLI. Et pour mettre en place ces outils, nous avons effectué une analyse et conception. L'analyse nous a permis de dégager les besoins fonctionnels et non fonctionnels, ainsi que les cas d'utilisation globaux de la plateforme. Alors que dans la conception, nous avons exposé les architectures physiques des environnements et de plateforme de livraison continue. Et enfin, nous avons terminé par une mise en place des différents outils.

Dans le cadre du projet AMC, nous avons appliqué l'infrastructure en tant que code avec Ansible pour gérer la configuration et le déploiement dans les environnements, qui a permis de réduire le temps de préparation des machines. De plus, nous avons regroupé les tâches Jenkins dans un pipeline en tant que code qui s'exécute à chaque changement de code source et déploie automatiquement dans les environnements d'intégration après la passation des tests.

Lors de ce travail, nous avons rencontré des problèmes liés à l'approvisionnement des machines qui ne peuvent pas être automatisées à cause de plateforme de virtualisation.

Comme perspective à ce travail, nous proposons l'intégration de l'automatisation des processus de monitoring applicatif de performances et des machines.

Bibliographie et Webographie

[1] https://www.technologies-ebusiness.com/enjeux-et-tendances/agilite-devops-combinaison-gagnante-partie-1 Le blog des technologies digitales décryptées par SQLI: La transformation digitale ne peut réussir que si elle s'inscrit dans une démarche globale d'adaptabilité et d'agilité dont DevOps est peut-être la solution, consulté en Mai 2018.

[2] https://www.cairn.info/revue-documentaliste-sciences-de-l-information-2010-3-page-54.htm Des outils pour toutes les plateformes, consulté en Mai 2018.

[3] https://assessment-tools.ca.com/tools/continuous-delivery-tools/fr/source-control-management-tools Définition de l'outil de gestion du code source, consulté en Avril 2018.

[4] https://assessment-tools.ca.com/tools/continuous-delivery-tools/fr/continuous-integration Définition de l'outil d'intégration continue, consulté en Avril 2018.

[5] http://www.pipobimbo.info/blog/techno/integration-continue/outil-de-qualimetrie/ Rôle outil de qualimétrie, consulté en Avril 2018.

[6] https://www.sonarqube.org/ Site officiel SonarQube, consulté en Avril 2018.

[7] https://scrutinizer-ci.com/ Site officiel Scrutinizer, consulté en Avril 2018.

[8] https://www.codacy.com/ Site officiel Codacy, consulté en Avril 2018.

[9] https://assessment-tools.ca.com/tools/continuous-delivery-tools/fr/artifact-repository-management Définition des outils de gestion des référentiels d'objets, consulté en Avril 2018.

[10] https://assessment-tools.ca.com/tools/continuous-delivery-tools/fr/configuration-management-tools Définition de l'outil de gestion de configuration, consulté en Avril 2018.

[11] https://wiki.hybris.com/display/hybrisALF/Deployment+Architecture Architecture déploiement Hybris, consulté en Juin 2018.

[12] https://jenkins.io/solutions/pipeline/ Site officiel Jenkins: Explication du Pipeline as Code, consulté en Juin 2018.

[13] https://docs.ansible.com/ansible/latest/dev_guide/overview_architecture.html Documentation Ansible 2.5 : Architecture Ansible, consulté en Juin 2018.

[14] https://go.cloudbees.com/docs/cloudbees-documentation/cookbook/book.html#_continuous_delivery_with_jenkins_pipeline Introduction Jenkins par CloudBees, consulté en Juin 2018.

[15] https://www.safaribooksonline.com/library/view/containerizing-continuous-delivery/9781491986851/ Livraison continue en Java, consulté en Juin 2018.

Mise en place d'Ansible dans un pipeline de livraison continue

Résumé

Afin de réduire le délai de commercialisation et produire des logiciels de qualité, l'entreprise SQLI opte à intégrer les pratiques DevOps dans les projets de ses clients. Dans ce cadre vient ce projet pour améliorer et optimiser un pipeline de livraison continue, qui gère les changements depuis le code source jusqu'à la livraison en production, en intégrant l'outil Ansible pour la gestion de configuration et l'automatisation de déploiement. Après une étude des principales valeurs et pratiques de DevOps notamment la livraison continue et la gestion de configuration, nous avons effectué une étude comparative des outils d'automatisation de ces pratiques. Suite à l'étude comparative, une analyse des besoins et une conception de l'architecture du projet se sont apparues nécessaires, avant la mise en place du pipeline de livraison continue optimisé. Ce projet a permis de minimiser le temps de déploiement dans les différents environnements et d'avoir une résilience contre les pannes d'infrastructures.

Mots clés : Agile, DevOps, Livraison Continue, Gestion de configuration, Ansible.

Setting up Ansible in a continuous delivery Pipeline

Abstract

In order to reduce the time to market and produce quality software, SQLI opts to integrate DevOps practices into its clients' projects. In this context, this project aims to improve and optimize a continuous delivery pipeline, which manages changes from source code to delivery of a release in production, with the integration of the tool Ansible, for configuration management and automation of deployments. After a study of the main values and practices of DevOps including continuous delivery and configuration management, we performed a comparative study of automation tools for these practices. Following the comparative study, an analysis of the different needs and a design of the project architecture appeared necessary, before the implementation of the optimized continuous delivery pipeline. This project allows to minimize the time of deployments in the different environments, and to have resilience against infrastructure failures.

Keywords: Agile, DevOps, Continuous Delivery, Configuration management, Ansible.

Master Systèmes Intelligents & Réseaux
Département d'Informatique
Faculté des Sciences et Techniques de Fès

A.U. 2017 - 2018