



REPUBLIQUE DU SENEGAL

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR DE LA RECHERCHE ET DE
L'INNOVATION



DAKAR AMERICAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
COMPUTER SCIENCE DEPARTMENT

MEMOIRE DE FIN DE CYCLE POUR L'OBTENTION DU DIPLOME D'INGENIEUR EN GENIE INFORMATIQUE

Titre :

Etude et mise en place d'une architecture d'automatisation du pipeline

DevSecOps

Année Universitaire 2021 - 2022

Soutenu le 07, janvier, 2023

Membres du jury :

Président : Professor Cheikh Sarr, Université Iba Der THIAM de Thiès

Superviseur: Mrs. Sokhar Samb, Assistante professor, Dakar American University of Science and Technology

Maître de stage : Mme. CISSÉ Rokhaya Dieye, Chef du département Recherche et Développement, Accel Technologie

Département de Génie Informatique

Dakar American University of Science and Technology / Somone, SENEGAL

Educating the next generation of Engineers, Inventors, and Innovators

www.daust.org / info@daust.org / [@daustofficial](https://twitter.com/daustofficial) / +221 33 898 8713 / +221 77 162 6223

DÉDICACES & REMERCIEMENTS

Mes remerciements s'adressent principalement à mes encadrants, Mme Rokhaya Dieye, Mlle Aminata Diallo, Mr Mohamed Fall et Mr Mame samba Ndiaye pour m'avoir soutenu tout au long de mes recherches. Avec des conseils, leurs disponibilités ainsi que leur compréhension dans ma façon de procéder. De même, je remercie mon Advisor a DAUST, Mlle Sokhar Samb, pour son suivi et son soutien au cours de ce trajet.

Je remercie aussi mes chers collègues pour les connaissances partagées durant cette expérience très riche d'enseignement. Je tiens à remercier tout particulièrement et à témoigner toute ma reconnaissance à toute ma famille, ma mère et mes frères, pour leur soutien pendant cette période de recherches.

Enfin, merci à toute l'équipe d'ACCEL TECHNOLOGIES, et mention spéciale à l'équipe R&D d'avoir rendu ce chemin beaucoup plus facile et agréable à traverser.

RESUME

Les logiciels et Internet ont transformé le monde et ses secteurs d'activité, du commerce au Divertissement en passant par les banques. Les logiciels ne se contentent plus de soutenir les entreprises : ils sont aujourd'hui un composant essentiel de leurs activités. Les entreprises interagissent Avec leurs clients à travers des logiciels livres en tant que services en ligne ou en tant qu'applications et sur toutes sortes d'appareils. Elles peuvent également utiliser les logiciels pour gagner efficacité opérationnelle en transformant chaque aspect de la chaine de valeurs, comme la logistique, les communications et les opérations. De la même manière que les entreprises travaillant avec des biens physiques ont transformé leurs méthodes de conception, de création et de livraison de produits à l'aide de l'automatisation industrielle tout au long du 20iem siècle, les sociétés d'aujourd'hui doivent adapter leur façon de créer et de livrer des logiciels.

La transition vers le DevOps implique un changement de culture et d'état d'esprit. Pour simplifier, le DevOps consiste à éliminer les obstacles entre deux _équipes traditionnellement isolées l'une de l'autre : l'équipe de développement et l'équipe d'exploitation. Certaines entreprises vont même Jusqu'à ne pas avoir d'équipes de développement et d'exploitation séparées, mais des ingénieurs assurant les deux rôles à la fois. Avec le DevOps, les deux équipes travaillent en collaboration pour optimiser la productivité des développeurs et la fiabilité des opérations. Elles cherchent à communiquer fréquemment, à gagner en efficacité et à améliorer la qualité des services offerts aux clients.

Elles assument l'entière responsabilité de leurs services, et vont généralement au-delà des rôles ou postes traditionnellement dénis en pensant aux besoins de l'utilisateur final et à comment les satisfaire. Les équipes d'assurance qualité et de sécurité peuvent également s'intégrer davantage à ces équipes. Les organisations adoptant un modèle axé sur le DevOps, quelle que soit leur

RESUME

structure organisationnelle, rassemblent des équipes qui considèrent tout le cycle de développement et d'infrastructure comme faisant partie de leurs responsabilités.

TABLE DES MATIÈRES

TABLE DES MATIÈRES

DÉDICACES & REMERCIEMENTS.....	i
RESUME	ii
TABLE DES MATIÈRES	iv
LISTE DES ACRONYMES.....	vii
TABLE DES FIGURES	viii
LISTE DES TABLEAUX	x
INTRODUCTION	1
1. CYCLES DE DEVELOPPEMENT	3
1.1. Cycles de développement traditionnel	3
A. Le modèle en cascade	3
B. Le modèle en V.....	6
1.2. Cycles de développement moderne.....	10
A. L'approche Agile	10
B. Les méthodes Agiles : Scrum	14
2. INTRODUCTION AUX TESTS ET A LA QUALITE LOGICIELLE	20
2.1. Les bugs et les conséquences.....	20
A. Qu'est-ce qu'un bug ?.....	20
B. Qualification de la gravité des anomalies	21
2.2. Introduction au test logiciel	21
A. Qu'est-ce que tester ?	22
B. Les principes de base de l'activité test	22
2.3. Introduction à la recette.....	24
A. Définition de la recette.....	24
B. Les étapes.....	25
C. Les documents livrables	27
2.4. Introduction a la Qualité logicielle	28
A. Fondements de la qualité logicielle.....	29
B. Processus de gestion de la qualité logicielle	30
C. Les considérations pratiques.....	32
3. DEVOPS : DE QUOI S'AGIT-IL ?.....	35
3.1. DevOps : les origines	35
3.2. Qu'est-ce que le DevOps ?	36

TABLE DES MATIÈRES

3.3.	Intégration continue.....	37
3.4.	Déploiement continu.....	39
3.5.	Déploiement continu ou Livraison continue ?	39
3.6.	DevOps et la sécurité.....	39
A.	Qu'est-ce que l'approche DevSecOps ?.....	41
4.	APPROCHE METHODOLOGIQUE.....	44
4.1.	Les outils de gestion de code source.....	44
A.	Périmètre d'action.....	45
B.	Bénéfices	45
C.	GitHub vs GitLab vs BitBucket	47
4.2.	Les outils de CI.....	48
A.	Périmètre d'action.....	49
B.	Bénéfices	49
C.	Bamboo vs Jenkins	50
4.3.	Les outils d'analyse de la qualité de code	51
A.	Périmètre d'action.....	52
B.	Bénéfices	52
C.	Sonarqube vs Veracode vs Codacy	52
4.4.	Les outils de gestion de configuration.....	54
A.	Périmètre d'action.....	55
B.	Bénéfices	55
C.	Puppet vs Chef vs Ansible.....	56
4.5.	Les outils de messagerie et de collaboration	58
A.	Pourquoi la collaboration est-elle cruciale pour DevOps ?	59
B.	Que signifie réellement la collaboration pour DevOps ?	60
C.	Slack et Trello	60
4.6.	Les outils de suivi de projet/d'anomalies.....	61
A.	Périmètre d'action.....	62
B.	Bénéfices	62
C.	Jira	63
4.7.	Les conteneurs	63
A.	Périmètre d'action.....	63
B.	Bénéfices	64
C.	Docker vs rkt from CoreOS vs LXC.....	64

TABLE DES MATIÈRES

4.8. Les outils d'orchestration de conteneurs.....	67
A. A quoi sert l'orchestration des conteneurs ?	68
B. Kubernetes vs OpenShift vs Nomad	69
5. MISE EN ŒUVRE	72
5.1. Architecture Continuous Deployment	72
5.2. Fonctionnement des différents éléments de l'architecture	73
A. Jenkins, pourquoi utiliser les pipelines ?	73
B. Docker, qu'est-ce que c'est ?.....	77
C. Le déploiement d'applications avec ansible ?	79
D. Pourquoi avons-nous besoin de Kubernetes ?.....	81
E. Pourquoi répéter le déploiement sur Openshift ?	85
5.3. Activités sécurité à forte valeur ajoutée intégrables dans un pipeline devops	85
A. Tests sécurité : tests d'intrusion automatisés	86
B. Tests sécurité : tests unitaires de scénarios d'abus	90
C. En quoi consiste la sécurité des conteneurs ?.....	90
5.4. Installation et configuration des solutions de l'architecture	95
A. Installation Jenkins	95
B. Installation Kubernetes	98
C. Installation Ansible	102
D. Installation SonarQube.....	103
CONCLUSION	109
BIBLIOGRAPHIE ET WEBOGRAPHIE	110

LISTE DES ACRONYMES

LISTE DES ACRONYMES

API	Application Programming Interface
AWS	Amazon Web Services
CD	Continuous Deployment/Delivery
CI	Continuous Integration
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MOA	Maitrise d'Ouvrage
MOE	Maitrise d'œuvre
OWASP	Open Web Application Security Project
QA	Quality Assurance
REST	Representational State Transfer
SaaS	Software as a Service
SCM	Source Code Management
SRE	Site Reliability Engineering
VA	Vérification d'Aptitude
VABE	Vérification d'Aptitude a la Bonne Exploitabilité
VABF	Vérification d'Aptitude au Bon Fonctionnement
VM	Virtual Machine
VSR	Vésication de Service Régulier

TABLE DES FIGURES

TABLE DES FIGURES

Figure 1-1: Processus de la méthode en cascade ou Waterfall	5
Figure 1-2: Cycle en V	7
Figure 1-3: Cycle en V pratique	9
Figure 1-4: Cycle en V pratique	10
Figure 1-5: Jalons d'un projet agile	14
Figure 1-6: Les 3 piliers de SCRUM.....	15
Figure 1-7: Processus Scrum.....	16
Figure 2-1: Processus du MBT	25
Figure 3-1: Processus Devops.....	36
Figure 4-1: Carte des principaux outils de CI/CD.....	44
Figure 5-1: Architecture Continuous Deployment	72
Figure 5-2: Exemple scénario de livraison continue avec les pipelines.....	75
Figure 5-3: Conteneurs traditionnels Linux vs Docker	78
Figure 5-4: Kubernetes	82
Figure 5-5: Aperçu du rôle de Kubernetes dans notre infrastructure.....	84
Figure 5-6: Les pratiques de sécurité recommandées par l'OWASP dans son modèle de maturité.....	86
Figure 5-7: Ajout du répertoire et installation de Jenkins	95
Figure 5-9: Activation du service de Jenkins	95
Figure 5-8: Interface Graphique de Jenkins	95
Figure 5-10: Intégration avec Sonarqube	96
Figure 5-11: Création de User	96
Figure 5-12: Intégration des serveur (ansible, dockerhost, et du master Node)	97
Figure 5-13: Ajouter des plugins d'intégration.....	98
Figure 5-14: Intégration des comptes à Utiliser	98
Figure 5-15: Préparation de l'environnement	98
Figure 5-16: Ajout des ports et des noeuds	99
Figure 5-17: Configuration du Sysctl	99
Figure 5-18: Activation des config.....	99
Figure 5-19: Ajout du répertoire et installation des paquets	100
Figure 5-21: Configuration du réseau.....	101
Figure 5-20: Configuration du système de Calico.....	101
Figure 5-22: Intégration des nœuds worker.....	101
Figure 5-23: Liste des nœuds	101
Figure 5-24: Installation de Python et Création de l'utilisateur	102
Figure 5-25: Configuration des Privilèges pour l'utilisateur	102
Figure 5-26: Connection avec les autres serveurs	102
Figure 5-27: Teste de L'installation et de la connection.....	103
Figure 5-28: Désactivation de SELinux.....	103
Figure 5-29: Configuration du sysctl.....	103
Figure 5-30: Création de l'utilisateur et Installation des paquets de Postgress.....	104
Figure 5-31: Vérification du service de Postgress.....	104
Figure 5-32: Connection avec le serveur host	105
Figure 5-34: Test de connexion avec la base de données.....	106
Figure 5-33: Téléchargement et Intégration de Sonarqube avec Java.....	106

TABLE DES FIGURES

Figure 5-35: Intégration de Sonarqube avec Postgress	107
Figure 5-36: Vérification du service de Sonarqube.....	107
Figure 5-37: Accès au Firewall	107
Figure 5-38: Interface Connexion à Sonarqube.....	108
Figure 5-39: Interface Graphique de Sonarqube	108

LISTE DES TABLEAUX

LISTE DES TABLEAUX

Tableau 2-1: Les différents niveaux d'anomalies	21
Tableau 4-1: Tableau comparatif GitHub - GitLab - BitBucket	48
Tableau 4-2: Tableau comparatif Jenkins - Bamboo	51
Tableau 4-3: Tableau comparatif Sonarqube - Veracode – Codacy	54
Tableau 4-4: Tableau comparatif Puppet - Chef - Ansible.....	58
Tableau 4-5: Tableau comparatif Docker - rkt - LXC.....	67
Tableau 4-6: Tableau comparatif Kubernetes - OpenShift - Nomad	71

INTRODUCTION

INTRODUCTION

Les entreprises traditionnelles divisent leurs équipes par type de travail. Par cette organisation, les équipes sont isolées l'une de l'autre. Certains départements s'occupent spécifiquement de la programmation. Plusieurs compagnies ont un département dédié au test des applications. En effet, selon eux, déployer un logiciel vers la production et l'entretenir demandent d'autres compétences que la programmation. Un département opérationnel a vu le jour. Chaque département est conduit par un manager spécialisé dans le domaine. Fractionner les différentes branches de telle manière semble contribuer à une bonne gestion, pourtant cela va générer des conflits.

Chaque département définit ses objectifs en fonction de la répartition du travail. Le département de développement peut être mesuré par sa vitesse dans l'implémentation de nouvelles fonctions, alors que le département opérationnel peut être jugé par la durée de disponibilité du serveur et le temps de réponse de l'application. Parfois, le travail des équipes opérationnelles est considéré comme réussi lorsque les paramètres sont stables et immuables. De l'autre côté, le développement est considéré productif lorsque l'équipe a opéré plusieurs changements. Cet engrenage rend la collaboration entre les deux groupes peu probable car il crée des conflits sur les motivations, le processus et les outils.

Les équipes de développement et opérationnelles ont chacune leur point de vue sur ce qu'est la performance :

- Les développeurs misent sur le besoin de changement : nouvelles fonctionnalités, corrections des bugs par exemple. L'équipe se base sur les demandes de changement. Ils s'attendent à ce que leur travail soit déployé sur l'application en production.

INTRODUCTION

- Les opérationnels ont peur du changement. Une fois le logiciel livré, l'équipe opérationnelle veut éviter que des changements se fassent sur le logiciel. Ce qui assurerait la stabilité des conditions des systèmes en production.

Une fois, le blocage entre développeurs et opérationnels dépasse et le logiciel mis en production, intervient un autre facteur : l'échec. L'échec est un facteur naturel pour le développement de n'importe quel système. Peu importe la quantité d'efforts que vous mettez, l'échec est toujours présent, et a un moment donné, il va se produire. Jusqu'à présent, il est rare de voir une entreprise se demander "que pouvons-nous faire s'il y a un bug à 4 heures du matin ?". Plus le code est complexe et mal documenté, plus il est difficile de localiser un bug. De même une énorme quantité de logs provenant des serveurs doivent être étudiés. Des bugs qui dépendent d'une combinaison de conditions imprévues et improbables sont particulièrement difficiles à localiser. Et plus l'entreprise met du temps pour gérer ces incidents plus il perd de l'argent. Il y va donc de la bonne maîtrise du budget. Il devient alors impératif d'étudier des méthodes et des processus qui permettent de gérer ces imprévus. Nous nous fixons comme objectifs :

- D'étudier des méthodes de cycles de développement ;
- D'introduire sommairement les tests et la qualité logicielle ;
- De définir DevOps et quelques pratiques clés pouvant aider à innover plus rapidement ;
- De proposer une architecture DevOps dans un environnement de développement.

1. CYCLES DE DEVELOPPEMENT

1.1. Cycles de développement traditionnel

Au fil des ans, les entreprises ont écarté le développement de leurs systèmes informatiques de leurs cœurs de métier. Cependant la réalité a rejoint très rapidement des entreprises telles qu'Amazon ou Alibaba, qui peuvent attribuer leur succès au maintien de leurs systèmes informatiques au cœur de l'entreprise.

Il y a quelques années, les entreprises avaient l'habitude d'externaliser l'ensemble de leurs systèmes informatiques, essayant d'éloigner la complexité de l'activité principale de la même manière que les entreprises externalisent la maintenance des bureaux où elles se trouvent. Cela a fonctionné pendant assez longtemps car les cycles de publication des mêmes applications ou systèmes étaient assez longs (quelques fois annuels) pour être en mesure d'articuler une chaîne complexe de gestion du changement car une sortie était un événement de type big bang où tout a été mesuré au millimètre avec peu ou pas de tolérance pour l'échec.

A. Le modèle en cascade

Traditionnellement, le cycle de développement et de livraison d'une application logicielle suit une liste d'étapes en ordre séquentiel. Cette façon de faire à l'avantage de fournir un plan de travail défini dès le départ, ce qui résulte la plupart du temps en une gestion budgétaire plus précise pour le client, mais pas nécessairement pour l'équipe de développement. Voici les étapes principales du processus de développement logiciel dans ce contexte :

1. Collecte et analyse des exigences : toutes les exigences possibles du système à développer sont saisies dans cette phase et documentées dans un document de spécification des exigences.

[Requirements]

CYCLES DE DEVELOPPEMENT

2. Conception du système : les spécifications des exigences de la première phase sont étudiées dans cette phase et la conception du système est préparée. Cette conception du système permet de spécifier les exigences en matière de matériel et de système et de définir l'architecture globale du système.

[Design]

3. Mise en œuvre : avec les données de la conception du système, le système est d'abord développé en petits programmes appelés unités, qui sont intégrés dans la phase suivante. Chaque unité est développée et testée pour sa fonctionnalité, ce qui est appelé test unitaire. **[Development]**

4. Intégration et test : toutes les unités développées dans la phase de mise en œuvre sont intégrées dans un système après le test de chaque unité. Après l'intégration, l'ensemble du système est testé pour détecter d'éventuels défauts et défaillances. **[Testing]**

5. Déploiement du système : une fois les tests fonctionnels et non fonctionnels terminés, le produit est déployé dans l'environnement du client ou mis sur le marché. **[Deployment]**

6. Maintenance : certains problèmes surviennent dans l'environnement du client. Pour résoudre ces problèmes, des correctifs sont publiés. De même, pour améliorer le produit, des versions améliorées sont publiées. La maintenance est effectuée pour apporter ces changements dans l'environnement du client. **[Maintenance]**

CYCLES DE DEVELOPPEMENT

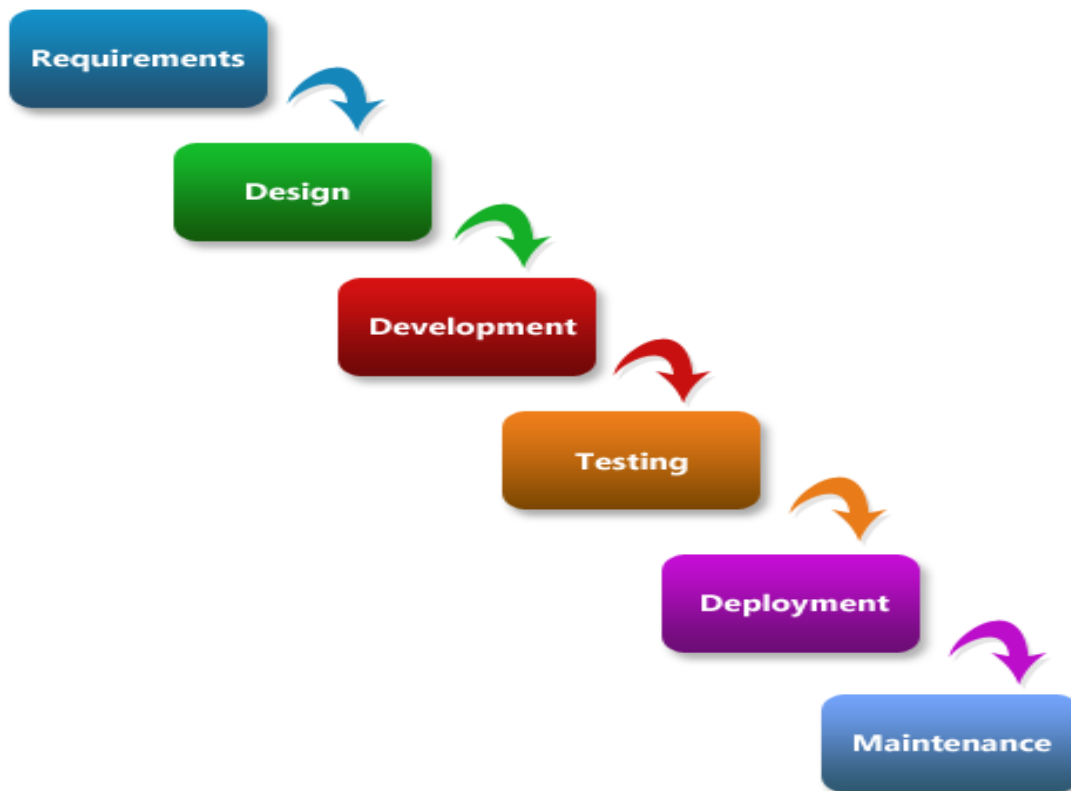


Figure 1-1: Processus de la méthode en cascade ou Waterfall

Typiquement, le processus recommence son cycle entre les étapes 4 à 6, jusqu'à ce que le logiciel soit livré au client. Suite à la livraison, chacune des mises à jour de développement suit à nouveau ces étapes. Ce modèle est plus connu sous le nom de « Waterfall model », dans lequel chaque composante du logiciel est développée séquentiellement, et menée à terme avant qu'une autre fonctionnalité ne soit programmée. C'est le modèle le plus souvent utilisé, mais il présente plusieurs lacunes importantes.

Problématiques

Les besoins du client

En général, l'analyse des besoins du client se fait conjointement avec lui, sur une période qui pourra varier considérablement et s'étendre parfois jusqu'à plusieurs semaines, voire plusieurs mois. Or, ce modèle Waterfall ne tient pas compte de la variable temps. Une fonctionnalité développée, testée puis livrée dans le logiciel arrivera à destination plusieurs

mois après qu'elle ait été dénie et pensée, pour répondre à un besoin au moment où l'analyse a été faite. Cependant, la réalité du client peut changer rapidement. Ce modèle de développement ne tient pas compte des besoins changeants du client, de sa réalité d'affaire qui est en mouvement au fil du temps, et des technologies émergentes. Le résultat est que la fonctionnalité est bien livrée selon les spécifications initialement documentées, mais ne correspond plus nécessairement aux besoins réels du client. Plus la durée de vie de l'application est longue, plus le code source de celui-ci se trouve alors alourdi par des fonctions et routines obsolètes et non utiles.

Masque de flexibilité

Dans ce modèle de cascade où une fonctionnalité ne peut pas être implémentée n'importe quand et dépend des fonctionnalités précédentes, il n'est pas possible de « changer son fusil d'épaule facilement ». C'est d'ailleurs la raison principale pour laquelle les développeurs ou designers qui travaillent sur le projet détestent autant les demandes de changements. Ce modèle de Waterfall ne cultive pas le changement. Au contraire, il prône l'immobilisme. L'application n'est souvent pas utilisable par le client tant que tout (ou presque) ne soit terminé à 100%. L'équipe ne dispose donc d'aucune souplesse dans la réalisation du projet, puisqu'avec un déroulement séquentiel, la réalisation d'une fonctionnalité dépend des réalisations précédentes. Il est donc particulièrement complexe de prendre en compte toute demande de changement, car elle aura un impact sur les développements courants et futurs, et nécessitera de revenir sur des fonctionnalités déjà en place, avec tous les êtes négatifs que cela peut engendrer. Le cauchemar pour n'importe quel programmeur !

B. Le modèle en V

CYCLES DE DEVELOPPEMENT

Le cycle en V est une méthode d'organisation très connue dont l'origine remonte à l'industrie et qui a été adaptée à l'informatique dans les années 80. La grande force du cycle en V, c'est qu'il définit assez précisément la manière dont les choses devraient se passer. Les phases de conception et de validation se découpent en plusieurs parties. Chaque étape ne peut être réalisée qu'une fois que l'étape précédente est terminée, ce qui diminue les prises de risque sur le projet.

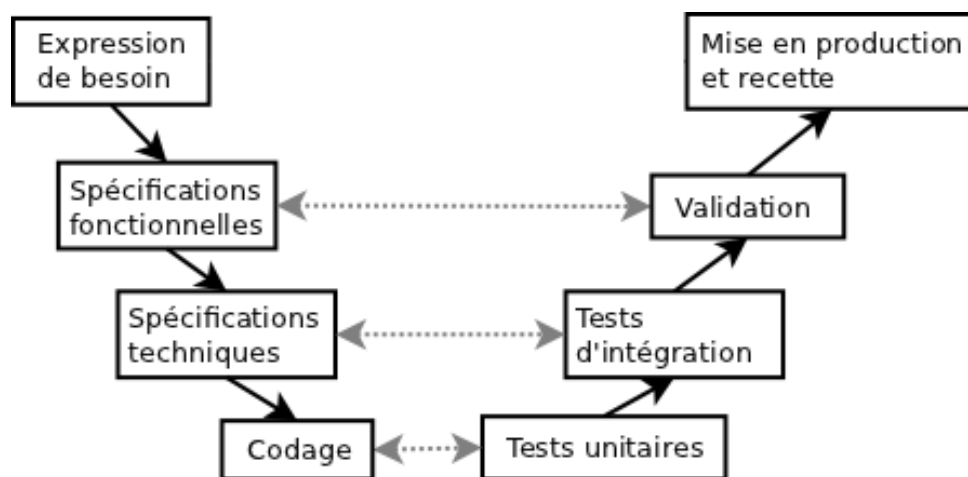


Figure 1-2: Cycle en V

On peut y distinguer 3 grandes parties :

- la phase de conception
- la phase de réalisation (codage)
- la phase de validation

Ce qui est bien visible sur le diagramme, c'est que chaque étape de conception possède son alter ego de validation. Il devient alors assez aisé de valider un projet, car le référentiel de test est connu très précisément.

Les différentes étapes

Le cycle en V est constitué de 8 étapes qui ont toute leur importance :

Expression de besoin :

CYCLES DE DEVELOPPEMENT

Le client exprime son besoin, en décrivant les usages correspondant au produit ni tel qu'il peut l'imaginer. Cela doit répondre aux questions « Que veut-on ? » Et « A quel cout ? ».

Spécifications fonctionnelles :

C'est le cahier des charges exact du produit final, tel que le désire le client. Il doit couvrir l'intégralité des cas d'utilisation du produit, en expliquant ce qu'il doit faire et non pas comment il va le faire.

Spécifications techniques :

C'est une traduction des spécifications fonctionnelles en termes techniques. C'est durant l'élaboration des spécifications techniques que sont choisies les technologies à mettre en œuvre pour développer le produit, et qu'est conçue l'architecture logicielle du produit.

Codage :

C'est la phase de réalisation à proprement parler, pendant laquelle sont développées des briques qui sont ensuite assemblées pour créer le produit fini.

Tests unitaires :

Ces tests interviennent à un niveau « atomique ». Chaque brique logicielle a été modélisée puis codée durant les étapes précédentes. Les tests unitaires assurent que ces briques respectent de manière individuelle leur cahier des charges.

Tests d'intégration :

Ce sont là les premiers tests grandeur nature du produit ni. On s'assure qu'il suit les indications des spécifications techniques.

Validation :

Le produit est à ce moment teste en regard de la spécification fonctionnelle. Toutes les utilisations qui y ont été déniées doivent pouvoir se vérifier dans les faits.

CYCLES DE DEVELOPPEMENT

Mise en production et recette :

Le produit est vérifié une dernière fois en pré-production, avant d'être mis en production.

Le client procède à la recette, pour vérifier que son expression de besoin est respectée.

La pratique

Malheureusement, si le cycle en V est limpide d'un point de vue théorique, son application réelle est très difficile. Dans une grande majorité de cas, on voit des organisations qui ressemblent plutôt à ce schéma :

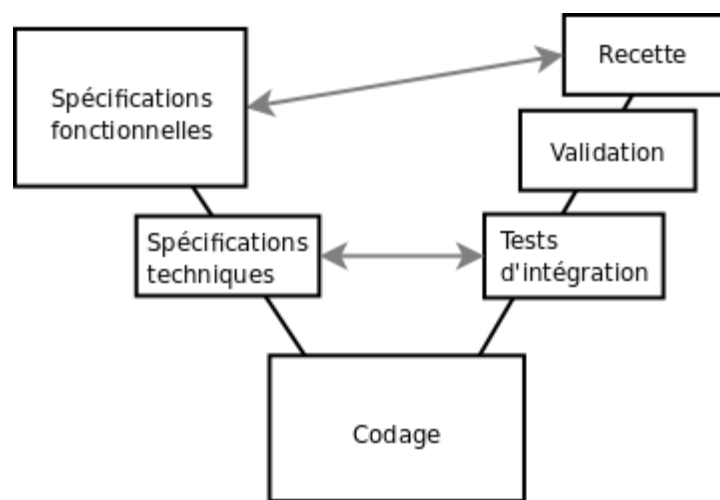


Figure 1-3: Cycle en V pratique

- La phase de conception se réduit à 2 étapes.
 - Les spécifications fonctionnelles, qui représentent l'ensemble des besoins du client et/ou définissent ce que doit faire le produit fini.
 - Les spécifications techniques, qui détaillent comment le produit va être réalisé techniquement.
- La phase de validation contient juste 3 étapes.
 - Les tests d'intégration, pendant lesquels on vérifie que l'intégralité du produit est valide techniquement.

CYCLES DE DEVELOPPEMENT

- Les tests de validation, qui sont un mélange de tests techniques et fonctionnels, et sur lesquels le client se base souvent pour décider du lancement du produit.
- La recette, qui est utilisée pour vérifier que le produit est valide par rapport aux spécifications fonctionnelles, mais qui a tendance à n'intervenir qu'après la mise en production (ou bien elle est tronquée en pré-production, ce qui aboutit à mettre des bugs en production).

Une telle organisation possède encore des avantages structurels assez convaincants, car elle définit toujours les différentes étapes de la réalisation d'un projet. Mais bien souvent, on se retrouve en réalité avec une organisation qui ressemble plutôt à quelque chose comme ça :

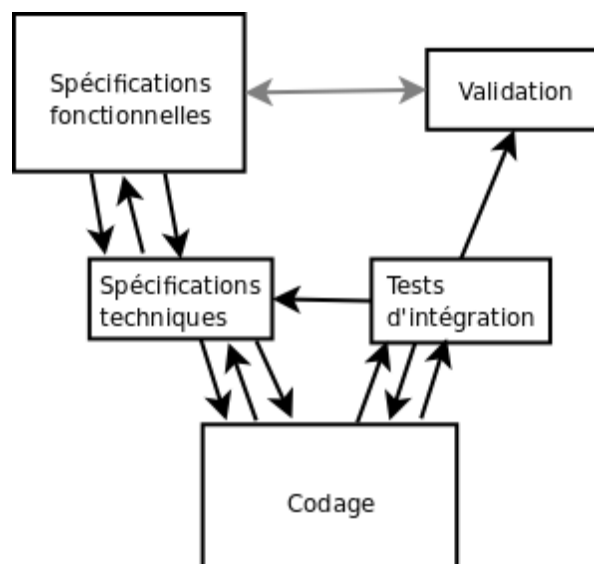


Figure 1-4: Cycle en V pratique

1.2. Cycles de développement moderne

A. L'approche Agile

Nous allons découvrir ce qu'est une approche agile. Et pour commencer, écartons ce que **N'EST PAS** une approche agile. Ce n'est malheureusement pas une baguette magique permettant de réussir un projet à coup sûr. En revanche, ça ne l'empêche pas d'offrir des leviers permettant de démultiplier ses chances de réussite, surtout si ce projet est complexe. Ce qui est souvent le cas

CYCLES DE DEVELOPPEMENT

puisque, on le constate, le simple fait de faire travailler des êtres humains ensemble, peut s'avérer complexe.

Alors qu'est-ce qu'une approche agile ? En fait, il s'agit d'une approche utilisant une ou plusieurs méthodes qui partagent des valeurs et principes communs formalisés en 2001 dans le « manifeste agile ». Ce manifeste nous dit :

Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. Ces expériences nous ont amenés à valoriser :

Les individus et leurs interactions plus que les processus et les outils

Des logiciels opérationnels plus qu'une documentation exhaustive

La collaboration avec les clients plus que la négociation contractuelle

L'adaptation au changement plus que le suivi d'un plan

Nous reconnaissons la valeur des seconds éléments, mais privilégions les premiers.

Ces 4 valeurs s'appuient sur les 12 principes suivants :

1. Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée ;
2. Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client ;
3. Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines _a quelques mois et une préférence pour les plus courts ;
4. Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet ;
5. Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont elles ont besoin et faites-leur confiance pour atteindre les objectifs fixés ;

CYCLES DE DEVELOPPEMENT

6. La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face
7. Un logiciel opérationnel est la principale mesure d'avancement ;
8. Les processus Agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant ;
9. Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité ;
10. La simplicité "c'est-à-dire l'art de minimiser la quantité de travail inutile" est essentielle ;
11. Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées ;
12. A intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Ce manifeste illustre finalement ce qu'on peut appeler l'état d'esprit agile. Car plus qu'une simple méthodologie, il s'agit avant tout d'un état d'esprit porteur de valeurs et de principes fondamentaux. Alors, comme vous avez pu le constater à la lecture du manifeste, l'origine des méthodes agiles provient du secteur du développement logiciel même s'il intéresse désormais d'autres secteurs. Ces méthodes sont nées d'un constat sur la réalité du terrain. Un constat qui consiste notamment à dire que les projets de développement logiciel sont devenus si complexes que l'incertitude est désormais inévitable sur de tels projets. Et pire que cela, pour un nouveau logiciel - et on pourrait presque dire un nouveau produit tout court - le besoin ne peut pas être complètement connu tant que les utilisateurs ne l'ont pas utilisé. Partant de ce constat, une poignée d'experts de terrain ont pris conscience qu'il était temps d'attaquer la gestion d'un projet

CYCLES DE DEVELOPPEMENT

sous un autre angle que celui de l'approche traditionnelle. En repartant d'une page blanche. C'est ainsi que plusieurs méthodes alternatives au cycle en V ont commencé à voir le jour. Des méthodes extrêmement pragmatiques, puisque créées et éprouvées par des personnes de terrain. Les méthodes agiles partent du principe que spécifier dans les détails l'intégralité d'un produit avant de réaliser ce dernier, et chercher à établir un plan détaillé du projet avant de se lancer, est contre-productif. L'idée consiste à se fixer un premier objectif _a courts-termes et se lancer dessus sans tarder. Une fois ce premier objectif atteint, on marque une courte pause et on adapte son itinéraire en fonction de la situation du moment. On peut au passage réaliser une projection de l'heure d'arrivée basée sur les mesures faites sur les tronçons déjà franchis. Une projection qui ne sera jamais exacte mais qui sera déjà plus _able que les prédictions de l'approche classique. Car cette projection sera basée sur l'expérience. On parle donc d'une approche empirique. Juste avant, il est intéressant de se demander pourquoi le fait de devenir agile devient essentiel aujourd'hui. Dans un monde soumis à une compétition acharnée, avec notamment le phénomène de mondialisation, la capacité à s'adapter rapidement et collectivement est essentielle à la survie d'une entreprise. Tout comme l'importance de prendre soin de ses collaborateurs pour pouvoir innover grâce à l'intelligence collective. Autrement dit, notre société a cruellement besoin de managers agiles. En résumé, un projet agile peut être représenté comme suit :

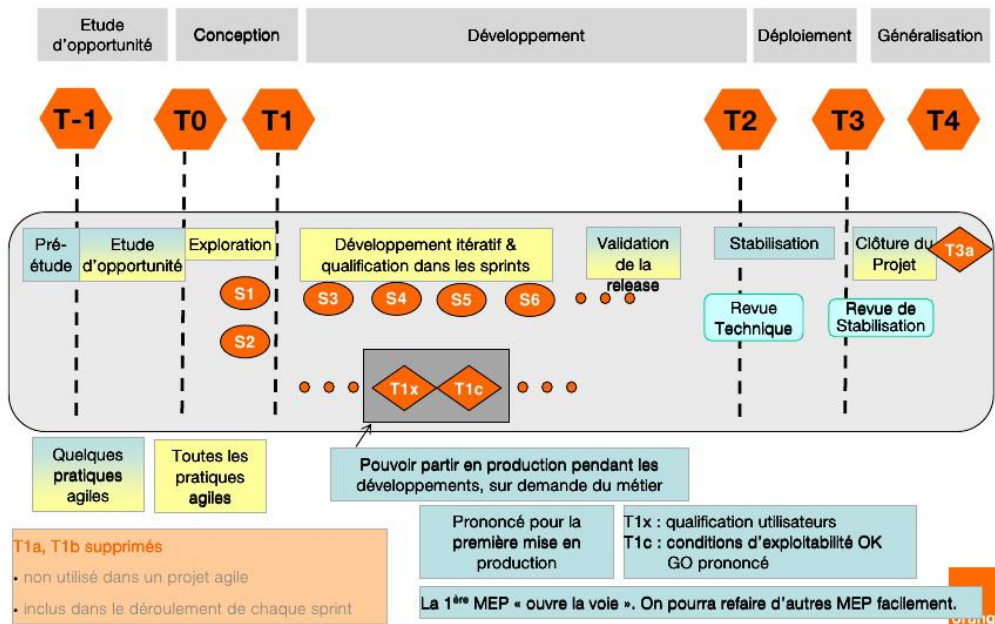


Figure 1-5: Jalons d'un projet agile

B. Les méthodes Agiles : Scrum

Parmi ces différentes méthodes, Scrum est de très loin LE plus utilisé. On dit LE parce qu'il s'agit d'un cadre méthodologique et non pas d'une méthode à proprement parlée. On pourrait presque résumer Scrum à la phrase « Diviser pour mieux maîtriser ». Dans le sens où on divise le périmètre à accomplir en sous-ensembles d'éléments porteurs de valeur. On divise le temps en intervalles courts appelés itérations, ou « sprints » pour utiliser le vocabulaire de Scrum, dans lesquelles on conçoit, réalise et teste de nouvelles fonctionnalités augmentant la valeur du produit. Et sur un gros projet, on divise les troupes en équipes de moins de 10 personnes pour une coordination et une efficacité optimale.

Fonctionnement de Scrum

CYCLES DE DEVELOPPEMENT

Nous allons voir le fonctionnement concret de Scrum en déroulant le processus associé. Mais juste avant, il est nécessaire de passer par un minimum de théorie. Face à la complexité d'un projet, Scrum oppose d'une part, la simplicité et légèreté méthodologique, et d'autre part, une approche empirique plutôt que prédictive.

Un processus empirique consiste à acquérir des connaissances provenant de l'expérience et apprendre des décisions basées sur des faits connus.

Et pour y parvenir, Scrum s'appuie sur 3 piliers fondamentaux :

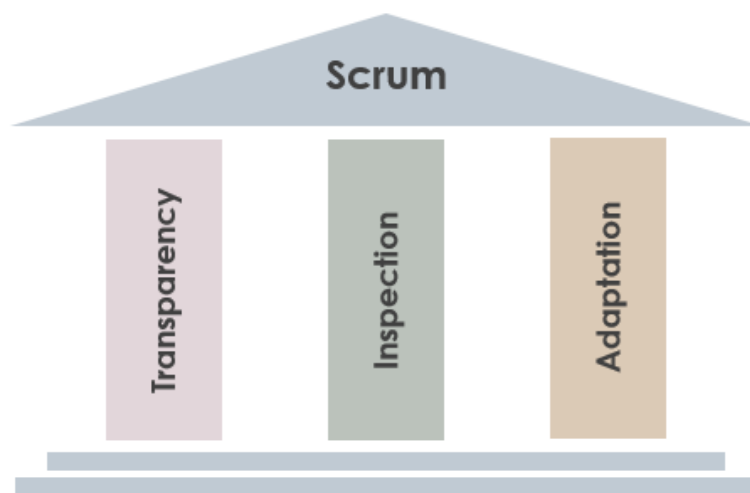


Figure 1-6: Les 3 piliers de SCRUM

La transparence qui permet de donner de la visibilité concrète sur le produit en train de se construire. Une visibilité donnée en particulier à ceux qui sont responsables du résultat du projet

L'inspection pour détecter les écarts indésirables par rapport aux objectifs.

L'adaptation pour rectifier ces écarts.

Il est à préciser que l'empirisme n'est pas antinomique avec la planification, au contraire.

Eisenhower¹ disait : ‘ ‘ j'ai toujours trouvé que les plans étaient inutiles, mais planifier est essentiel ’ ’.

¹ Dwight David Eisenhower est le 34^{ème} président des Etats-Unis

CYCLES DE DEVELOPPEMENT

Ce que l'on peut facilement comprendre quand on pense aux imprévus que comporte une bataille. Etablir un plan à l'avance et planifier régulièrement, sont bien deux choses différentes.

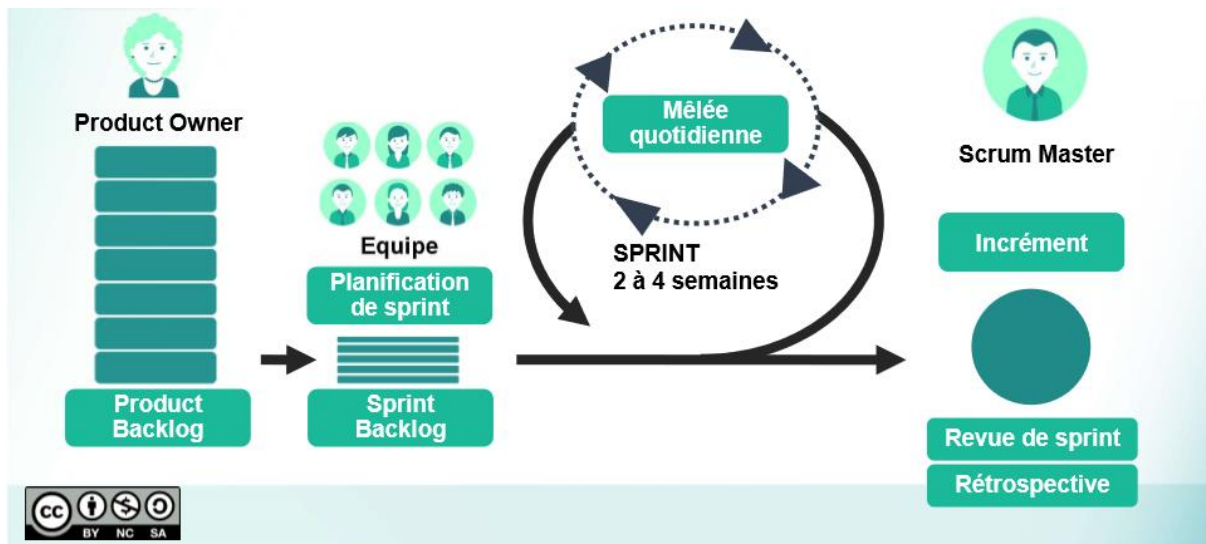


Figure 1-7: Processus Scrum

Parcourons ensemble le processus Scrum. Tout commence avec le **Product Owner** qui porte la vision du produit que l'on souhaite réaliser et représente les futurs utilisateurs de ce dernier. Dans une entreprise classique, c'est généralement un acteur de la maîtrise d'ouvrage. Ici c'est une femme, appelons la AWA.

Après avoir travaillé sur la vision du produit - qui consiste notamment à répondre à des questions du genre :

- Pourquoi fait-on ce produit ?
- Quel problème va-t-il résoudre ?
- Qui va l'utiliser ?

AWA centralise dans ce qu'on appelle le Product Backlog l'ensemble de ses besoins. Symbolisées par les gros pavés verts. Ils sont gros puisqu'à ce stade, on ne rentre pas dans la conception détaillée, on reste à un niveau macroscopique. AWA sollicite également l'équipe de développement pour estimer chaque pavé individuellement et compléter le Product Backlog par

CYCLES DE DEVELOPPEMENT

des éléments qui lui auraient échappé, purement techniques par exemple. Pour précision, l'équipe de développement a pour rôle de réaliser le produit.

Une fois qu'elle dispose de toutes les estimations, elle va pouvoir ordonnancer son Product Backlog en s'efforçant de mettre en haut de ce dernier, des éléments qui apportent une forte valeur ajoutée au produit ou considérés comme indispensables à une première version de ce dernier. Et en bas, des éléments moins importants, qui sont généralement des éléments à faible valeur ajoutée et coûteux. L'ordonnancement n'est pas un exercice facile, donc elle s'appuie sur d'autres acteurs, comme l'équipe de développement ou d'autres parties prenantes du projet.

Maintenant que le Product Backlog est prêt et qu'on a convenu de la durée des itérations, on peut démarrer le sprint² en procédant à la réunion de planification de ce dernier. Cette réunion consiste tout simplement à sélectionner un sous ensemble d'éléments du Product Backlog qui seront réalisés dans le cadre du sprint en cours. Cette réunion réunit principalement AWA et l'équipe de développement. L'équipe de développement détermine la quantité d'éléments à sélectionner en fonction de sa capacité à faire. Une capacité à faire qui est difficile à évaluer lors des premiers sprints mais qui sera de plus en plus _able par la suite, car l'équipe pourra alors se baser sur l'expérience des sprints précédents. Les éléments sélectionnés se retrouvent dans ce qu'on appelle le **Sprint Backlog** qui détermine finalement le périmètre du sprint et qui contient également le plan d'action destiné à atteindre l'objectif fonctionnel du sprint. Le Sprint Backlog peut faire l'objet d'un découpage en sous-tâches des éléments sélectionnés, ce qui peut permettre de répartir plus facilement le travail au sein de l'équipe de développement et faciliter aussi la mesure de l'avancement au quotidien. Comme toutes les réunions Scrum, la

² Sprint étant synonyme d'itération pour Scrum

CYCLES DE DEVELOPPEMENT

planification de sprint est timeboxée³. Sa durée maximale est de 8 heures pour des sprints d'un mois et elle est généralement plus courte pour des sprints plus courts.

Une fois la planification de sprint terminée, les travaux de réalisation commencent, incluant la conception détaillée, mise en œuvre et tests des fonctionnalités. Ces travaux sont pilotés par les fonctionnalités en soi et non pas par des phases. Autrement dit, on ne reproduit pas un mini cycle en V en sein du sprint. Chaque jour, au cours de la mêlée quotidienne, l'équipe de développement se synchronise et inspecte son avancement vis-à-vis de l'objectif du sprint. Elle planifie ses actions pour les prochaines 24 heures. Cette réunion est limitée à 15 minutes. Et pour respecter ce timing, on reste debout et on n'instruit pas les problèmes en séance mais plutôt après la mêlée avec les personnes concernées. Le sprint, lui aussi est timeboxée. Il se termine par les réunions successives de revue de sprint puis de rétrospective de sprint. Lors de la revue de sprint, le produit incrémente de nouvelles fonctionnalités est présente aux parties prenantes. Dans le cadre du premier sprint, il s'agit bien sûr des premières fonctionnalités. Parmi les parties prenantes, on peut avoir les sponsors du projet, un panel d'utilisateurs dont les retours seront très utiles, ainsi que le service marketing ou l'équipe de conduite du changement selon le type de projet. Lors de cette revue de sprint, on fait également le point sur l'avancement global du projet et on décide des éventuelles nouvelles orientations à prendre en fonction de cet avancement, des retours formules lors de la démonstration de l'incrément et du contexte du moment.

Enfin la rétrospective de sprint quant à elle consiste à tirer profit de l'expérience vécue sur le sprint écoule pour améliorer l'efficacité du processus de réalisation dès le sprint suivant. Cette réunion limitée à 3 heures pour des sprints d'un mois et généralement moins pour des sprints plus courts, rassemble toute l'équipe Scrum. Autrement dit, AWA, l'équipe de développement

³ Limitée dans le temps

CYCLES DE DEVELOPPEMENT

et le Scrum Master. C'est un peu le « post-mortem » du sprint, ce bilan qu'on est censée faire à la fin d'un projet traditionnel pour capitaliser pour le projet suivant. Une bonne pratique rarement utilisée, peut-être en raison de ses limites. Puisque toutes les leçons que l'on pourrait tirer de ce post-mortem ne servent plus à rien pour le projet qui vient de se terminer. Et d'autre part, le projet suivant sera certainement différent. Au contraire, sur un projet Scrum, on tire pleinement parti des leçons apprises au cours de chaque sprint. Et ce, dès le début du projet. Ensuite, on boucle en procédant à la planification du sprint suivant et ainsi de suite. Au fil des sprints, le produit s'enrichit. Le Scrum Master quant à lui est le garant du respect de Scrum. Il s'assure que tout le monde adhère à Scrum et s'approprie son rôle.

2. INTRODUCTION AUX TESTS ET A LA QUALITE LOGICIELLE

2.1. Les bugs et les conséquences

Le problème fondamental du développement des logiciels est le problème de l'erreur. De nombreux défauts et anomalies peuvent affecter un logiciel. En effet, nous porterons notre attention sur les sources des bugs logiciels et les niveaux de la gravité des anomalies.

A. Qu'est-ce qu'un bug ?

Est-ce si grave ? D'où vient l'erreur ? Quelles sont les conséquences désastreuses ? Y'a-t-il un moyen de chasser les bugs ? Le terme BUG est utilisé pour désigner soit :

- Un défaut
- Une erreur
- Une anomalie

Défaut

Une imperfection dans un composant ou un système qui peut conduire à ce qu'un composant ou un système n'exécute pas les fonctions requises, par exemple une instruction ou une définition de données incorrecte. Un défaut, si rencontré lors de l'exécution, peut causer la défaillance d'un composant ou d'un système.

Erreur

Action humaine produisant un résultat incorrect.

Anomalie

Toute condition qui dévie des attentes basées sur les exigences de spécifications, documents de conception, documents utilisateurs, standards, etc., ou des perceptions ou expériences de quelqu'un. Les anomalies peuvent être trouvées pendant, mais pas uniquement,

INTRODUCTION AUX TESTS ET A LA QUALITE LOGICIELLE

les revues, tests, analyses, compilations ou utilisation des produits logiciels ou de la documentation applicable.

B. Qualification de la gravité des anomalies

Il existe plusieurs niveaux d'anomalies illustrés sur ce tableau ci-dessous :

Gravite	Description
Bloquante	L'anomalie provoque un arrêt complet du système ou une fonctionnalité indispensable est inexploitable (pour des raisons fonctionnelles ou de performance, aucune solution) de contournement. Une anomalie bloquante doit faire l'objet d'un correctif rapide.
Majeure	Une fonctionnalité indispensable est partiellement inopérante sans bloquer l'exploitation de l'outil. L'application peut continuer. Une solution de contournement est identifiée, qu'il ne faut pas négliger.
Mineure	Une fonctionnalité non essentielle présente des dysfonctionnements sans bloquer l'exploitation de l'outil. La solution peut attendre et les dysfonctionnements seront traités en dernier.

Tableau 2-1: Les différents niveaux d'anomalies

2.2. Introduction au test logiciel

Toute fabrication de produit suit les étapes suivantes :

- Conception
- Réalisation
- Test

Le test est un sujet vaste et complexe : les applications sont variées, les outils sont différents, cependant, les tests doivent être organisés et menés selon une démarche.

A. Qu'est-ce que tester ?

Il existe plusieurs définitions sur l'activité de test logiciel. Cependant on peut prendre cette définition : le test d'un logiciel est une activité qui fait partie du processus de développement. Il est mené selon les règles de l'assurance de la qualité et débuté une fois que l'activité de programmation est terminée. Il s'intéresse aussi bien au code source qu'au comportement du logiciel. Son objectif consiste à minimiser les chances d'apparitions d'une anomalie avec des moyens automatiques ou manuels qui visent à détecter aussi bien les diverses anomalies possibles que les éventuels défauts qui les provoqueraient.

B. Les principes de base de l'activité test

Principe N°1 : Impossible de tout tester

Dans une application informatique, il n'est pas possible de tout tester. Cependant, avant de tester une application, il faut bien s'assurer que les tests couvrent les exigences fonctionnelles et techniques également. Il a été démontré scientifiquement, que la preuve du zéro défaut est un problème indécidable. Il est donc impossible de garantir l'absence de défaut dans les logiciels. Cependant il est important de définir une stratégie de test et analyser les risques.

Principe N°2 : Mise en évidence de défauts

Pour tester correctement un logiciel il faut des connaissances du contexte, de l'intelligence et de la créativité, il ne faut jamais croire que c'est simple et qu'il n'y a pas d'erreurs. Si aucun défaut n'est découvert, ce n'est pas une preuve qu'il n'en reste pas.

Principe N°3 : Tester le plus tôt possible

La planification sérieuse des tests dans le planning est indispensable à la maîtrise du projet, cela nous permet d'effectuer les tests le plus tôt possible dans le cycle de développement du logiciel. Il a été prouvé que des vérifications et des corrections effectuées au moment des exigences sont moins coûteuses que celles effectuées au moment du déploiement de l'application. « Les études menées sur le coût associé à la détection d'une erreur montre que si une erreur décelée lors de la phase d'élaboration du cahier des charges coûte 1 alors la même erreur décelée en phase de conception coûte 10 et une erreur décelée en phase d'exploitation coûte 100 ».

Principe N°4 : Les tests dirigés par les risques

Analyser les risques, et les points de vigilances de l'activité test nous permet d'être attentifs aux enjeux économiques de l'application à tester. Il est indispensable de définir la priorité de chaque fonctionnalité et prendre en compte le dysfonctionnement et son éventuel impact sur l'organisation cible.

Principe N°5 : Assurance qualité (QA) logiciel

La présence d'un tiers de confiance (QA logiciel) est également indispensable pour garantir la qualité du logiciel. Le développement n'est pas le seul responsable et juge de l'évaluation honnête de la qualité du logiciel : « L'assurance qualité permet de mettre en œuvre un ensemble de dispositions qui vont être prises tout au long des différentes phases de fabrication d'un logiciel pour accroître les chances d'obtenir un logiciel qui corresponde à ses objectifs (son cahier des charges). »

2.3. Introduction à la recette

A. Définition de la recette

C'est une opération formelle, réalisée par le client, pour vérifier la conformité du produit fourni (composants logiciels spécifiques, progiciels, prestations et documentation) par rapport à la spécification validée (plus, éventuellement, les modifications acceptées). Cette vérification se fait sur la base d'un dossier de recette et se termine par un procès-verbal de recette qui formalise l'acceptation du client.

Une étape très importante dans un projet informatique, mais très délicate ; il arrive souvent des tensions entre le maître d'œuvre et le maître d'ouvrage et les testeurs. Parfois la MOA propose des solutions techniques pour la MOE alors que ce n'est pas son rôle, parfois la MOE intervient sur les choix fonctionnels, et surtout reproche à la MOA son implication superflue en pensant que projet informatique est une affaire des informaticiens. Des problèmes sont souvent le fait de la méconnaissance de l'objet de la recette et des techniques associées. Le schéma ci-dessous, représente le Model-Based Testing (MBT) : piloter la production et la maintenance des tests fonctionnels et d'acceptation à partir d'une modélisation de l'expression de besoins : processus et règles métier.

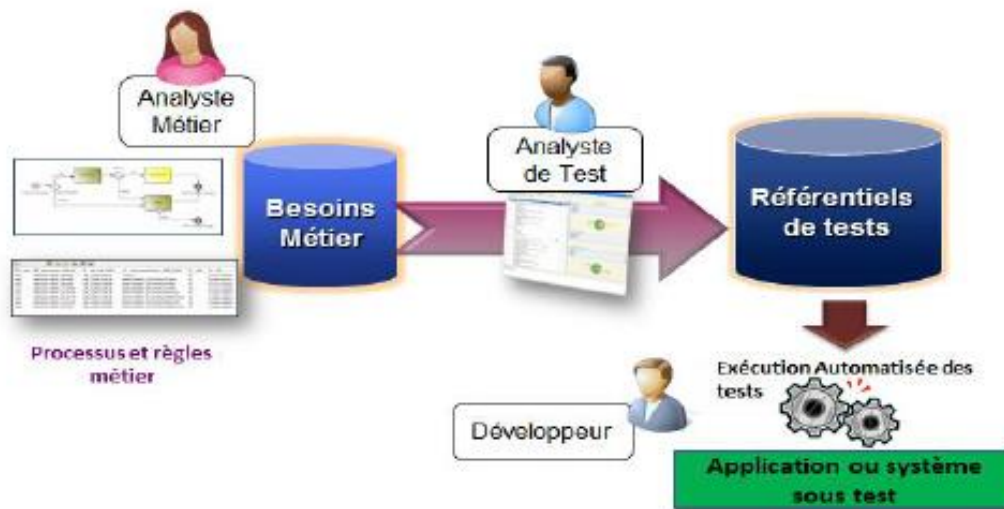


Figure 2-1: Processus du MBT

B. Les étapes

La procédure de recette se déroule en deux étapes principales :

1. Les tests système ;
2. Les tests d'acceptation utilisateur.

Si la première étape a lieu chez le fournisseur, la deuxième se déroule en revanche généralement dans les locaux et avec les infrastructures du client.

La recette usine

La recette usine comprend tous les tests réalisés chez le fournisseur avant la livraison. Elle désigne donc les tests unitaires, les tests de validation et les tests d'intégration. D'un point de vue maîtrise d'ouvrage, la recette usine correspond à une période de quelques jours pour valider que la livraison correspondra à sa commande. En ce qui concerne les tests unitaires, de validation et d'intégration, ceux-ci sont considérés comme relevant des tests de qualification en amont, au niveau du fournisseur. Phase très importante pour celui-ci, car c'est elle qui lui permet

de vérifier que son produit est de qualité et de faire accepter au client l'installation sur ses plateformes des tests et recettes pour les tests utilisateur métier VA, puis pour la VSR. Par extension, elle signifie aussi la période (généralement courte) durant laquelle le client procède lui-même à ses propres tests dans ses locaux, avant d'accepter les livrables. A l'issue de la recette usine, le fournisseur et le client signent un procès-verbal de fin de recette usine, qui accompagne la livraison du produit et le cahier de recette.

La recette utilisateur ou VA ou VABF

Lors de l'étape de VA ou VABF (aptitude à répondre aux besoins exprimés dans le cahier des charges initial) ou recette utilisateur, le client réalise deux catégories de tests différentes. D'un côté, une recette technique est effectuée afin de vérifier que le produit livré est techniquement conforme sur toute la chaîne de processus. De l'autre, la maîtrise d'ouvrage contrôle l'aspect fonctionnel du produit lors de la recette fonctionnelle.

La recette fonctionnelle

La recette fonctionnelle a pour but la validation des fonctionnalités exprimées dans le cahier des charges et détaillées dans les spécifications fonctionnelles. La MOA procède donc à sa propre série de tests de validation.

La recette technique ou VABE

Chargée de contrôler les caractéristiques techniques du produit livré, la recette technique, ou VABE regroupe les tests suivants :

- Les tests d'exploitabilité : les tests de supervision, de sauvegarde... et en particulier les tests de respect des exigences d'architecture technique ;
- les tests de performance.

La VSR

Si la VABF se déroule correctement et est validée, le client procède alors à la mise en service opérationnelle. Une période de VSR commence donc par un premier déploiement sur un site pilote. Cette mise en production permet de valider le produit en conditions réelles. A la différence des étapes précédentes, celle-ci se déroule pleinement en environnement de production avec des données réelles.

C. Les documents livrables

Plusieurs documents accompagnent la procédure de recette :

Le protocole de recette ou stratégie de recette

Le protocole de recette est un document visant à clarifier intégralement la procédure de recette. Il précise scrupuleusement :

- Les tâches du client ;
- Les tâches du fournisseur ;
- La liste des documents à communiquer ;
- L'ordre des tests et le planning ;
- Les seuils d'acceptation du produit.

Le cahier de recette

Le cahier de recette est la liste exhaustive de tous les tests pratiqués par le fournisseur avant la livraison du produit. La couverture des tests, en particulier ceux de non-régression lorsqu'il s'agit d'une nouvelle version d'un produit existant, pouvant être infinie, le cahier de recette doit préciser toutes les fiches de test passées par le fournisseur, ainsi que celles à passer dans l'environnement du client lors de la VABF.

Les fiches de faits techniques

INTRODUCTION AUX TESTS ET A LA QUALITE LOGICIELLE

Les fiches de faits techniques visent à formaliser les écarts constatés en recette et sont classifiées d'un commun accord entre fournisseur et client en : anomalies et évolutions.

- Les anomalies, ou bugs, décrivent un écart du produit livré par rapport au comportement spécifié et attendu ; elles sont généralement issues d'une défaillance du fournisseur, et peuvent donner lieu à de futures corrections.
- Les évolutions correspondent à un écart du produit livré par rapport au comportement attendu ; elles sont généralement issues d'une défaillance ou lacune du client dans son expression de besoin, et peuvent donner lieu à des avenants ou un futur contrat.

Les procès-verbaux

Pour clore chaque étape de la procédure de recette, un procès-verbal est rédigé. Celui-ci a pour objet de prononcer la réception et de mentionner les réserves émises par chacune des parties. Toutes ces recettes peuvent être contractuelles et donner lieu à un procès-verbal de recette qui permet de prononcer la réception, assortie ou non de réserves. Ainsi le PV de recette usine conditionne le démarrage de la période de VA, celui de VABF celle de VSR.

2.4. Introduction a la Qualité logicielle

L'assurance qualité logicielle (AQL) est un ensemble d'activités planifiées et systématiques de toutes les actions nécessaires pour fournir une assurance suffisante qu'un logiciel produit ou modifié est conforme aux exigences et aux attentes établies. Les pratiques d'AQL varient selon le modèle d'affaire et l'industrie où le logiciel est utilisé.

Un corpus de connaissances vise à corriger les concepts essentiels d'une profession afin de normaliser sa pratique. Il contient les connaissances en AQL que tout développeur et ingénieur logiciel devraient avoir acquis. L'ISO 15939 (aussi nommée le guide SWEBOK) a pour objectif de promouvoir non seulement une vision cohérente du génie logiciel dans le monde entier, mais

aussi de fournir une base commune de connaissances pour l'élaboration de programmes de formation et de certification en génie logiciel. Le domaine du logiciel souffre encore d'un certain nombre de problèmes qui ont été résolus dans les autres domaines du génie : faible qualité, absence de garantie, dépassements des coûts et faible utilisation de normes. En introduction, l'ISO 15939 présente les considérations de qualité qui surpassent les processus individuels du cycle de vie d'un projet logiciel. On y précise que la qualité doit être un souci omniprésent pour l'ingénieur logiciel, et conséquemment il est important de toujours la considérer dans tous les autres domaines de connaissances présentés dans le guide, par exemple : les exigences, la conception, la maintenance, etc. Au chapitre onze du guide SWEBOK, on y décrit les connaissances généralement acceptées qui permettent de définir, d'instaurer et de mesurer la qualité d'un logiciel. Ces connaissances sont réparties en trois grandes catégories :

- Les fondements de la qualité logicielle ;
- Les processus de gestion de la qualité logicielle ;
- Les considérations pratiques.

A. Fondements de la qualité logicielle

Fondements de la qualité logicielle

Au cours des années, les auteurs et les organismes ont défini le terme « qualité » de manières différentes :

- « Se conformer aux exigences des utilisateurs »
- « une adéquation parfaite entre le logiciel et son utilisation visée »
- « La qualité telle qu'elle est définie par la clientèle »

- « La qualité exigée par le client »

La qualité du logiciel a aussi sa définition propre que l'on peut retrouver dans la norme internationale ISO 9001 : « le degré auquel un ensemble de caractéristiques remplit les exigences ». Toutes ces définitions sont générales. Dans certains cas, le contexte requiert que la qualité logicielle soit modélisée et compilée dans le dossier des exigences du logiciel. Plusieurs modèles de la qualité logicielle ont été proposés au cours des ans afin de décrire les exigences qualité d'un logiciel.

Il est important pour le spécialiste du logiciel de spécifier les exigences qualité, les communiquer et s'assurer que tous les intervenants approuvent ces exigences au tout début d'un projet ou d'une modification d'un logiciel. Le spécialiste en assurance qualité logicielle a pour mission de former et d'appuyer le personnel du développement, de la maintenance et des infrastructures afin qu'ils appliquent concrètement ces concepts au cours de leur travail quotidien. C'est à l'étape de l'élucidation des exigences d'un projet logiciel que les exigences qualité doivent être identifiées, documentées, numérotées, hiérarchisées et approuvées. Il est important de souligner ici que ces exigences ont une incidence sur les processus, techniques et outils de mesures d'assurance qualité logicielle qui devront être mise en place pour évaluer la qualité du logiciel livré à la clientèle.

Cette approche permet de démontrer, à la clientèle, que ces exigences ont été prises en compte et validées, par des tests, avant l'acceptation finale du logiciel.

B. Processus de gestion de la qualité logicielle

La gestion de la qualité logicielle s'applique dans toutes les perspectives des processus, des produits et des ressources IT. Elle définit les processus, les propriétaires de processus, les exigences pour la réalisation de ces processus, les mesures de processus et des sorties, ainsi que les canaux de rétroaction. Les processus de gestion de la qualité du logiciel se composent de

nombreuses activités. Certaines peuvent être utilisées pour trouver des défauts, alors que d'autres, comme les analyses causales, indiquent où il pourrait être avantageux d'analyser plus en profondeur pour prévenir des défauts. La planification de la qualité d'un logiciel implique donc :

- De définir les exigences et spécifications en termes de caractéristiques particulières de qualité d'un logiciel ;
- De planifier les processus et activités de qualité qui seront nécessaires pour s'assurer de rencontrer ces exigences autant en prévenant qu'en cherchant à trouver et éliminer le plus grand nombre de défauts tout en respectant l'enveloppe budgétaire et le calendrier.

La gestion de la qualité logicielle est définie dans la norme ISO 12207 [ISO 08a]. Cette norme Précise la portée de la gestion de la qualité du logiciel et décrit les processus suivants :

- L'assurance qualité logicielle (AQL) ;
- La vérification et validation (V&V) ;
- la revue et audit;
- La résolution des problèmes.

Ces processus favorisent la culture qualité et permettent également la mise en place d'activités de prévention de défauts. Les processus d'assurance qualité logicielle (AQL) offrent la possibilité d'assurer une meilleure qualité du logiciel dans la mise en place d'un projet de développement, de maintenance ou d'un projet de changement dans les infrastructures IT. Ils fournissent également, comme sous-produits, des informations synthétisées pour la prise de décision, y compris une indication de la qualité des processus logiciels de l'organisme. Les processus d'AQL permettent aussi de donner une indication concernant l'état réel de l'avancement des travaux (par exemple, les plans de management, le développement, la gestion

de configuration, l'analyse d'impact). Ils visent à évaluer si les produits intermédiaires et finaux répondent à leurs exigences initiales spécifiées.

Les résultats des activités d'AQL peuvent être synthétisées dans des tableaux de bord (technique qui permet de présenter des mesures clés de la performance d'un projet logiciel) pour la prise de décision et l'identification d'actions correctives. Le gestionnaire de l'AQL doit s'assurer que les résultats de ces rapports qualités sont vérifiés et pertinents.

Les processus d'AQL sont étroitement liés aux activités de cycle de vie du logiciel ; ils peuvent se superposer et parfois être inclus dans une activité du cycle de vie. Parfois, ils peuvent sembler réactifs parce qu'ils s'adressent à l'identification de défauts sur les processus tels qu'ils sont mis en pratique et les produits tels qu'ils sont développés. Afin qu'ils soient proactifs, les processus d'AQL doivent aussi être identifiés au tout début de l'étape de planification initiale de projet de développement ou de maintenance. Il a été démontré qu'en étant proactif, on pourra atteindre les caractéristiques de qualité et le degré de qualité, envisagés à moindres coûts.

C. Les considérations pratiques

L'ingénieur logiciel doit aussi avoir des connaissances pratiques en ce qui a trait à l'utilisation des principes de la qualité et des techniques courantes dans le domaine concerné. Cette section évalue les connaissances nécessaires pour vérifier si la qualité visée pourra être atteinte.

Les exigences de l'application de la qualité : Des facteurs influencent la planification, la gestion et le choix des activités et des techniques d'AQL lors du démarrage d'un projet de développement, de maintenance et d'opération, incluant :

- Le domaine du système dans lequel le logiciel opérera (sûreté-critique, sécurité-critique, mission-critique, affaires-critique, environnement-critique) ;

- Les exigences de système et de logiciel ;
- Les composants (internes), commerciaux (externes) ou standard à employer dans le système ;
- Les normes spécifiques de génie logiciel applicables à l'interne et imposées par le domaine concerne ;
- Les outils logiciels et méthodes a employer pour le développement et la maintenance ainsi que l'évaluation et l'amélioration de la qualité ;
- Le budget, le personnel, l'organisation du projet, les plans et l'établissement de l'échéancier de tous les processus ;
- Les utilisateurs et l'utilisation prévus du système ;
- Le niveau d'intégrité du système ;
- la taille de organisation;
- La répartition géographique des équipes de développement ;
- la diversité Culturelle.

L'information acquise en étudiant chaque facteur indique comment les processus du système de management de la qualité doivent être organisés et documentes. Cette information indique aussi quelles ressources seront nécessaires et les limites qui seront imposées dans la réalisation du projet.

Dans les cas où l'échec du système pourrait avoir des conséquences extrêmement graves, la fiabilité globale (matériel, logiciel, et humain) est une exigence de qualité (exigence non fonctionnelle) principale au-delà de la fonctionnalité requise par les utilisateurs. La fiabilité du logiciel inclut des caractéristiques telles que : la tolérance aux fautes, la sureté, la sécurité et « l'utilisabilité ».

INTRODUCTION AUX TESTS ET A LA QUALITE LOGICIELLE

Etant donné que le développeur ou l'ingénieur logiciel pourra avoir des préoccupations plus générales qui englobent le système (matériel, logiciel d'exploitation, télécommunications), le futur corpus de connaissances de l'ingénierie des systèmes (system engineering) s'assurera que ceux-ci doivent être très _ables (« confiance élevée » ou « systèmes de très haute intégrité »). Cette terminologie qui est utilisée couramment pour les systèmes mécaniques et électriques traditionnels (qui peuvent ne pas inclure de logiciel) a été adaptée pour traiter des menaces ou des catastrophes imprévisibles, des risques, de l'intégrité du système. L'ingénieur logiciel qui travaille dans un domaine critique devra vérifier que ses connaissances sont adéquates afin de faire face à des requis de qualité provenant de l'ingénierie des systèmes.

Le niveau d'intégrité d'un logiciel est basé sur les conséquences possibles de l'échec du logiciel et de la probabilité de cet échec. Pour le logiciel dans lequel la sûreté ou la sécurité est importante, des techniques telles que l'analyse de risques pour la sûreté ou l'analyse de menaces pour la sécurité peuvent être employées pour développer une activité de planification qui identifierait où se trouvent les points sensibles potentiels. L'historique des échecs de logiciels semblables peut également aider en identifiant quelles techniques seront les plus utiles pour détecter des défauts et en évaluer la qualité. On propose alors des niveaux d'intégrité pour le logiciel (par exemple la gradation d'intégrité) selon la norme IEEE 1012 pour le système critique.

3. DEVOPS : DE QUOI S'AGIT-IL ?

3.1. DevOps : les origines

Au cours des dernières années, nous avons traversé une révolution en IT : elle déferlerait des entreprises de IT pures à tous les secteurs : commerce de détail, banque, finance, etc. Cela a conduit à un certain nombre de petites entreprises appelées startups, généralement à la recherche d'importantes levées de fonds d'investissement, avec très fort potentiel éventuel de croissance économique, et de spéculation financière sur sa valeur future (création d'entreprise), à viser le marché mondial. Les entreprises telles que Amazon ou Alibaba, sans parler de Google, Apple, Stripe⁴ ou même Spotify, sont passées du garage de l'un des propriétaires à de grandes entreprises employant des milliers de personnes.

Une chose en commun dans l'étincelle initiale avec ces entreprises a toujours été l'inefficacité des grandes entreprises : plus l'entreprise est grande, plus il faut de temps pour accomplir de simples tâches.

Ce phénomène crée lui-même un marché, avec une demande qui ne peut être satisfaite des produits traditionnels. Afin de fournir un service plus agile, ces startups doivent être rentables. Il est tout à fait normal pour une grande banque de dépenser des millions sur sa plateforme d'échange de devises, mais si vous êtes une petite entreprise ou startup qui veut faire du chemin, votre seule possibilité contre une grande banque est de réduire les coûts par l'automatisation et de meilleurs processus. C'est une obligation pour les petites entreprises d'adopter de meilleures façons de faire les choses, car chaque jour qui passe est un jour plus proche du manque d'argent. Mais il y a une plus grande obligation pour adopter des outils DevOps : l'échec.

⁴ Société américaine d'origine irlandaise, destinée au paiement par internet pour particuliers et professionnels

DEVOPS : DE QUOI S'AGIT-IL ?

L'échec est un facteur naturel pour le développement de n'importe quel système. Peu importe la quantité d'efforts que vous mettez, l'échec est toujours présent, et à un moment donné, il va se produire.

Habituellement, les entreprises sont très concentrées sur l'élimination de l'échec, mais il y a une règle non écrite qui est de les empêcher : **la règle 80-20** ou la loi de **Pareto**.

Il faut 20% de temps pour atteindre 80% de vos objectifs. Les 20% restants prendront 80% de votre temps.

Passer une énorme quantité de temps à éviter l'échec est voué à l'échec.

3.2. Qu'est-ce que le DevOps ?

Le DevOps est une combinaison de philosophies culturelles, de pratiques et d'outils qui améliore la capacité d'une entreprise à livrer des applications et des services à un rythme élevé, pour des produits qui évoluent et s'améliorent plus rapidement que ceux des entreprises utilisant des processus traditionnels de développement de logiciel et de gestion d'infrastructure. Cette vitesse permet aux entreprises de mieux servir leurs clients et de gagner en compétitivité.

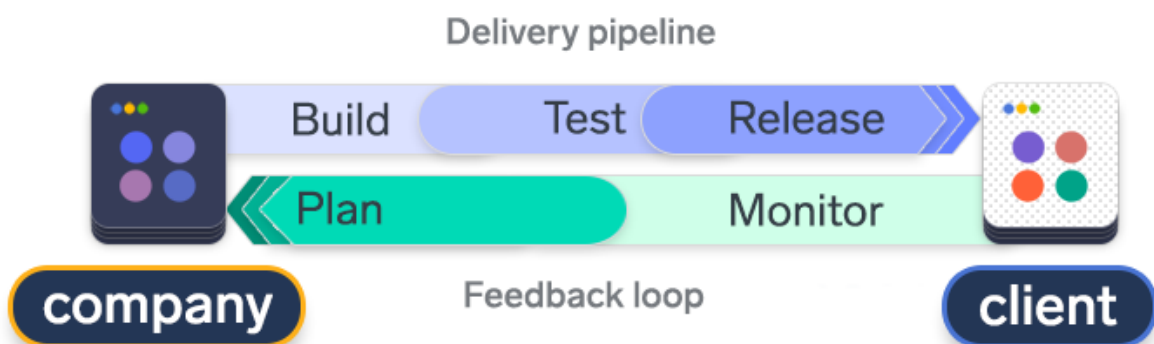


Figure 3-1: Processus Devops

DEVOPS : DE QUOI S'AGIT-IL ?

Dans un modèle axé sur le DevOps, les équipes de développement et d'exploitation ne sont plus isolées. Il arrive qu'elles soient fusionnées en une seule et même équipe. Les ingénieurs qui la composent travaillent alors sur tout le cycle de vie d'une application, de sa conception et son test jusqu'au déploiement et son exploitation, et développent toute une gamme de compétences liées à différentes fonctions. Les équipes d'assurance qualité et de sécurité peuvent également s'intégrer davantage au développement, à l'exploitation et au reste du cycle de vie d'une application. Ces équipes utilisent des pratiques pour automatiser des processus qui étaient autrefois manuels et lents. Elles exploitent un stack technologique et des outils qui les aident à opérer et à faire évoluer des applications de façon rapide et agile. Ces outils aident également les ingénieurs à accomplir de façon autonome des tâches (par exemple, le déploiement de code ou la mise en service d'infrastructure) qui nécessiteraient normalement l'aide d'autres équipes, ce qui augmente encore davantage leur productivité.

3.3. Intégration continue

L'intégration continue est une méthode de développement de logiciel DevOps avec laquelle les développeurs intègrent régulièrement leurs modifications de code à un référentiel centralisé, suite à quoi des opérations de création et de test sont automatiquement menées. L'intégration continue désigne souvent l'étape de création ou d'intégration du processus de publication de logiciel, et implique un aspect automatisé (un service d'IC ou de création) et un aspect culturel (apprendre à intégrer fréquemment). Les principaux objectifs de l'intégration continue sont de trouver et de corriger plus rapidement les bugs, d'améliorer la qualité des logiciels et de réduire le temps nécessaire pour valider et publier de nouvelles mises à jour de logiciels.

Autrefois, les développeurs au sein d'une même équipe avaient tendance à travailler séparément pendant de longues périodes et à n'intégrer leurs modifications au référentiel

centralisé qu'après avoir fini de travailler. L'intégration de tous ces changements cumulés était alors longue et difficile.

A cette contrainte s'ajoutait le problème de l'accumulation de petits bugs sur une longue période sans les corriger. La combinaison de ces différents facteurs empêchait de livrer rapidement des mises à jour aux clients.

Avec l'intégration continue, les développeurs appliquent régulièrement leurs modifications sur un référentiel partagé, avec un système de contrôle des versions. Avant d'envoyer leur code, les développeurs peuvent choisir d'exécuter des tests sur des unités locales pour le vérifier davantage avant son l'intégration. Un service d'intégration continue détecte les modifications faites sur le référentiel partagé. Il crée et exécute alors automatiquement des tests d'unité avec les nouveaux changements de code, afin de détecter instantanément les erreurs de fonction ou d'intégration.

Avantages de l'intégration continue

Améliorer la productivité des développeurs

L'intégration continue aide votre équipe à gagner en productivité, en limitant de nombre de tâches manuelles devant être accomplies par les développeurs et en encourageant les comportements qui contribuent à réduire le nombre d'erreurs et de bogues dans les versions publiées auprès des clients.

Trouver et corriger plus rapidement les bugs

Avec des tests plus fréquents, votre équipe peut découvrir et corriger plus rapidement les bogues avant qu'ils ne prennent de l'ampleur.

Livrer plus rapidement des mises à jour

L'intégration continue aide votre équipe à livrer plus rapidement et plus fréquemment des mises à jour après des clients.

3.4. Déploiement continu

Le problème commun rencontré par la plupart des développeurs est comment libérer le code implémenté rapidement et en toute sécurité. Le processus de livraison utilisé traditionnellement, cependant, est une source de pièges et conduit généralement à la déception des développeurs et des clients.

3.5. Déploiement continu ou Livraison continue ?

La livraison continue d'application ou de modifications (continuous delivery) et le déploiement continu (continuous deployment) constituent deux des piliers de la démarche DevOps. Et doivent évidemment s'articuler autour d'un retour continu des utilisateurs finaux. Néanmoins, il convient de distinguer ces deux notions.

Avec une méthode de Continuous Delivery, les développeurs produisent du code applicatif sur des cycles courts, en s'assurant de pouvoir le modifier ou le mettre à jour à tout moment (idéalement, sans arrêter la production). En optimisant ou automatisant le développement, les tests et la mise à disposition du code, les équipes réduisent fortement les délais et les coûts, tout en procurant à l'entreprise une informatique agile, réactive, voire proactive. Cependant, si le code à disposition est déployable en un ou deux clics, il n'est pas forcément automatiquement déployé.

En revanche, dans le cadre du déploiement continu, chaque modification est automatiquement déployée si les phases de tests (automatisés ou non) sont positives. Le déploiement continu nécessite donc un niveau supérieur de maturité et de maîtrise.

3.6. DevOps et la sécurité

Les initiatives DevOps sont de plus en plus répandues que l'entreprise soit une startup ou une multinationale. Dans la lignée des méthodologies agiles qui s'appuyaient entre autres sur des outils d'intégration continue, DevOps pousse plus loin ces pratiques en adoptant le déploiement continu et en faisant collaborer étroitement les équipes de développement et d'exploitation.

DEVOPS : DE QUOI S'AGIT-IL ?

Mais la sécurité ne fait que rarement partie de ces programmes, le management étant déjà bien occupé par ce changement d'organisation. Comme une initiative DevOps implique un contrôle sur la phase de développement, nous nous concentrerons dans cet article sur les entreprises qui produisent leur code source.

Historiquement, la sécurité est dans les mains des équipes d'exploitation, avec un fort accent sur les infrastructures réseau. Le principe fondateur est de tenter de conserver l'attaquant à l'extérieur du périmètre. Pour cela, toute une panoplie de solutions est couramment utilisée : firewall, reverse proxy, IDS, etc. Un autre aspect important est la réduction de la surface d'attaque avec un travail de configuration fine sur les systèmes d'exploitation et bases de données : désactivation des fonctions inutiles, mises à jour, droits d'accès, logs d'audit, chiffrement, etc.

En parallèle de cela, la sécurité applicative est trop peu souvent prise en compte dans le cycle de développement logiciel. Cela peut s'expliquer par l'organisation de l'entreprise, le responsable sécurité (CISO) ayant peu d'influence sur les équipes de développement et pas forcément d'expérience pratique dans ce domaine vu son parcours professionnel. Il doit donc considérer le logiciel comme une boîte noire et peut seulement émettre des recommandations de codage très génériques.

Il va alors essentiellement collaborer avec les équipes d'exploitation et leur fournir un budget pour investir dans des produits sécurité reconnus afin de « boucher les trous » des applications. Ces équipements nécessitent des étapes d'intégration supplémentaires, ce qui ralentit le déploiement. Mais cela cause aussi des problèmes en production, seul cet environnement ayant en général les règles les plus strictes configurées. Ces incidents sont souvent délicats à traiter par les équipes d'exploitation parce qu'ils nécessitent la connaissance détaillée de

DEVOPS : DE QUOI S'AGIT-IL ?

l'implémentation de l'application. On se retrouve alors dans une situation que cherche absolument à éviter une initiative

DevOps. Si on ajoute à cela la pression du métier, retenant surtout l'aspect déploiement accéléré, on peut rapidement se retrouver avec le niveau de sécurité minimum acceptable : démontrer que l'organisation se soucie en quelque sorte de la sécurité le jour où les choses tournent mal.

A. Qu'est-ce que l'approche DevSecOps ?

L'approche DevSecOps (Development - Security - Operations) est une approche qui permet d'intégrer la sécurité des données dès le début d'un projet. La sécurité de celles-ci est considérée comme une condition préalable avant de commencer. L'objectif est de pouvoir l'intégrer dans l'ensemble du cycle de vie du projet, du développement à la mise en œuvre, en utilisant des méthodes flexibles et l'approche DevOps. Intégré dans le processus DevOps, DevSecOps vous permet de configurer l'automatisation des tests de sécurité à chaque fois que votre application, logiciel, site internet est testé et déployé. Cela permet de détecter et de corriger rapidement les anomalies. L'approche DevSecOps implique de penser à la sécurité des applications et des infrastructures de votre projet dès sa conception. Il est également recommandé d'automatiser certaines tâches concernant la sécurité des données pour éviter de ralentir le déroulement des opérations de DevOps. Pour atteindre ces objectifs, il est nécessaire de commencer par sélectionner des outils qui peuvent assurer l'intégration continue des questions de sécurité, par exemple, avec un environnement de développement intégré commun qui fournit des fonctionnalités de sécurité. Cependant, pour que la sécurité DevOps soit efficace, cela nécessite plus que de nouveaux outils. Il est nécessaire d'intégrer culturellement les nouvelles procédures de sécurité au sein de vos équipes afin que cela soit mis en œuvre et appliqué le plus rapidement possible.

B. Quels sont les avantages de l'approche DevSecOps ?

DEVOPS : DE QUOI S'AGIT-IL ?

A. Définir et mettre en œuvre la sécurité à tous les niveaux et pour tous les environnements.

La sécurité doit être intégrée dès le début d'un projet (protection des données dès la conception) et dans les processus de développement et de mise en œuvre. De la même manière que le DevOps permet d'automatiser les déploiements applicatifs, le DevSecOps permet d'automatiser les tests de sécurité. Ceux-ci doivent être effectués immédiatement après la première livraison de votre projet (application, logiciel...). Cela s'applique aux environnements de développement, de démo, de test et de préproduction. La surveillance automatisée du résultat de ces tests alertera en temps réel les équipes concernées sur les vulnérabilités bien avant que le produit final n'atteigne l'environnement de production. Ce processus permet de détecter rapidement les lacunes et les écarts en matière de sécurité des données. Cela vous permet de réagir rapidement afin de pouvoir déployer un produit final qui aura bien moins de problèmes liés à la confidentialité et la sécurité des données.

B. Développer une culture de la sécurité auprès de vos collaborateurs

Pour que l'approche DevSecOps soit efficace, une coopération étroite est nécessaire entre les devs Ops, les équipes opérationnelles, le service informatique et les parties prenantes du projet. Par conséquent, afin de développer une culture de la sécurité, il est nécessaire de créer un climat de confiance entre les groupes. Afin que chacun se sente impliqué dans la coopération, des formations et des sessions de sensibilisation à la sécurité peuvent être planifiées. Le DSI joue un rôle important car il a une vue d'ensemble du projet et des équipes et permet de jouer un rôle transverse, nécessaire au bon fonctionnement du projet.

C. Utilisation des outils d'automatisation de la sécurité informatique

DEVOPS : DE QUOI S'AGIT-IL ?

Un processus DevSecOps efficace n'est possible qu'avec les bons outils. L'utilisation d'un scanner de vulnérabilité, par exemple, permet une analyse automatique des applications web. Ce type d'outil peut très bien s'intégrer dans vos processus opérationnels actuels ou dans vos nouveaux projets. L'intégration d'outils automatisés permet également de réduire les erreurs humaines lors des tests et du déploiement de la sécurité.

4. APPROCHE METHODOLOGIQUE



Figure 4-1: Carte des principaux outils de CI/CD

4.1. Les 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. 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'option aux développeurs travaillant sur des projets complexes. De plus, le SCM peut protéger les

fichiers source de toute sorte d'anomalie, et permet à toutes les équipes de savoir qui a effectué quel changement et à quel stade.

A. Périmètre d'action

Les outils SCM peuvent être conçus pour s'exécuter dans les environnements Open Source ou professionnels, et offrent aux utilisateurs une palette de fonctionnalités permettant de suivre les lignes de code et d'y apporter des modifications. Un élément clé, plus ou moins universel pour tous les produits disponibles, est un portail de gestion centralisé permettant aux membres d'une équipe de collaborer de façon fluide sans, par exemple, écraser les changements de chacun. En fonction de l'outil, le SCM peut aussi gérer des requêtes en mode « pull », des traitements basés sur des fonctionnalités, les permissions de branches, les discussions en ligne et l'intégration avec un grand nombre de différentes technologies.

B. Bénéfices

Les bénéfices immédiats pour les utilisateurs de SCM sont transparence, visibilité et contrôle.

Il faut également noter que de nombreux outils sont très élastiques et peuvent ainsi évoluer en même temps que les projets. La nature du SCM et ses fonctionnalités amélioreront la vitesse et la qualité de toute mise à jour ou livraison. Il permet à différents développeurs, à des endroits différents, de travailler sur le même projet, et fournit une traçabilité détaillée de qui a fait quoi. Un historique aussi détaillé permet d'identifier les responsabilités de chacun et sécurise tout projet, tout en laissant les développeurs libres d'expérimenter et de suivre les résultats.

Les systèmes de contrôle de version (ou référentiels) peuvent être de trois types principaux :

Local: Tous les développeurs sont dans le même système de fichiers.

Centralisé : Les équipes de projet possèdent une copie unique du projet sur un serveur central et les membres de l'équipe peuvent modifier un exemplaire.

Distribué : les développeurs travaillent dans leurs référentiels locaux et les modifications sont partagées entre les référentiels.

APPROCHE METHODOLOGIQUE

Nous allons parler expressément des dépôts Git. Les dépôts Git sont les plus efficaces pour les équipes de développement. Tout d'abord, découvrons quels composants doivent être présents dans un bon dépôt.

- **Pull Request** Lorsque les utilisateurs apportent des modifications au code source et le repoussent dans le référentiel, leurs collaborateurs sont notifiés pour obtenir les modifications discutées et examinées.
- **Code review** basé sur le Web
- **Modification** Si un référentiel suggère une édition collaborative et en temps réel, cela ajoute beaucoup à la qualité d'un repo.
- **Suivi des bugs** Aucun projet n'est créé sans bugs. C'est donc génial quand un référentiel permet de suivre et de résoudre des bugs en collaboration.
- Prise en charge de la syntaxe markdown pour rendre le code lisible.
- Authentification à deux facteurs pour assurer la protection des comptes d'utilisateurs.
- La possibilité de créer des forks⁵ ou des clones de référentiels.
- Des extraits qui permettent aux utilisateurs de partager des segments de code ou des fichiers avec les personnes dont ils ont besoin des membres de l'équipe, d'autres utilisateurs ou même eux-mêmes.
- Intégration avec les services tiers.
- Importation de référentiels Lorsque les utilisateurs passent d'un service à un autre, il est conseillé aux référentiels de leur permettre d'importer des projets existants.

⁵ Un fork est un nouveau logiciel créé à partir du code source d'un logiciel existant lorsque les droits accordés par les auteurs le permettent : ils doivent autoriser l'utilisation, la modification et la redistribution du code source. C'est pour cette raison que les forks se produisent facilement dans le domaine des logiciels libres.

- Licence open source dans certains cas. Parfois, les entreprises ont besoin d'avoir un référentiel interne sur leur propre serveur au lieu d'utiliser les ressources Web publiques existantes.

Il existe principalement 3 référentiels Git utilisés par les développeurs du monde entier.

C. GitHub vs GitLab vs BitBucket

GitHub

GitHub est le site Web numéro un pour le stockage des dépôts git et le développement de projets conjoints. Le système est conçu pour permettre aux utilisateurs de créer facilement des systèmes de contrôle de version spéciaux basés sur Git. Pourquoi est-ce tellement populaire ? Git prend en charge la fusion et la séparation de versions fluides à l'aide d'outils de visualisation et d'outils de navigation dans l'historique de développement non linéaire. Pour l'instant, GitHub héberge plus de 50 millions de projets open source.

GitLab

Le service est également développé sur la base du contrôle de version Git. Malgré le fait que la fonctionnalité GitLab soit similaire à son principal concurrent - GitHub, il y a quelques particularités majeures. GitLab peut avoir plusieurs formes différentes telles que GitLab SAAS qui convient aux entreprises et GitLab Community Edition, une solution individuelle pour les utilisateurs.

Bitbucket

Le service est également très similaire à GitHub et reflète la plupart de ses fonctionnalités avec de légères différences. Bitbucket est mieux orienté vers les équipes de développement professionnel car il leur fournit des conditions avantageuses telles que des repos privés gratuits, des intégrations Jira, une révision de code avancée et un CI/CD intégré. Parallèlement à la croissance de l'équipe, Bitbucket propose des conditions tarifaires plus

APPROCHE METHODOLOGIQUE

avantageuses comparables à GitHub et GitLab. Bitbucket fournit également un modèle de déploiement flexible pour les équipes.

Comparaison GitHub, GitLab, BitBucket


Version Control Comparison	 GitHub	 GitLab	 Bitbucket
Free Private Repos		✓	✓
Free Public Repos	✓	✓	✓
Merge Request / Issue Templates	✓	✓	
Integrated CI	*	✓	✓
Enterprise Plans	✓	✓	✓
Integrated Project Board	✓	✓	✓
Navigation Usability	✓	✓	
Open Source		✓	
Self-Hosted Option	✓	✓	✓
Discoverability	✓		
AD/LDAP	✓	✓	✓
LFS File Storage	✓	✓	✓
Integrated Time Tracking	*	✓	
Integrated Review Apps	*	✓	✓
Free, Public Static Web Pages	✓	✓	✓
Burndown Charts, Project Analytics		✓	✓
Third Party tool integration	✓	✓	✓

Tableau 4-1: Tableau comparatif GitHub - GitLab - BitBucket

4.2. Les outils de CI

APPROCHE METHODOLOGIQUE

Les logiciels de 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 bug soit détecté rapidement et transféré au développeur pour résolution. 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 et ou/équipes de travailler en parallèle sur différents aspects du même projet.

Alors que de plus en plus d'entreprises développent leurs équipes travaillant sur le CI/CD, il existe une demande croissante pour trouver la meilleure solution pour répondre à leurs besoins de déploiement.

A. Périmètre d'action

Les outils de CI offrent un grand nombre de fonctionnalités et certains sont plus appropriés que d'autres selon la situation, en fonction de leur compatibilité avec différents langages de programmation, versions, systèmes de contrôle et référentiels. En plus d'automatiser de nombreuses tâches manuelles et chronophages, CI fournit un historique détaillé du développement d'un projet. Ainsi, les programmeurs peuvent identifier plus rapidement ce qu'il s'est passé. Les tests identifient toute erreur sur le code, à la fois en termes de fonctionnalité et d'intégration à un plus grand projet.

De plus, dans certaines circonstances, CI constitue également le socle du processus de livraison continue, où l'intégralité du processus est automatisée {pas seulement du développement au test, mais jusqu'à la simulation et le processus de livraison.

B. Bénéfices

Les bénéfices de CI sont nombreux, et viennent de ce que les outils peuvent détecter des bugs tôt dans le processus de développement, ce qui facilite la mise en œuvre des changements appropriés, rapidement et efficacement. Cela réduit naturellement les coûts, permet d'accélérer le cycle de vie des livraisons en évitant les écarts entre différentes intégrations, et garantit la

livraison du projet a temps. Par nature, le CI améliore visibilité et collaboration entre différentes équipes de développement. En effet, les outils de CI permettent d'éviter un processus de livraison chaotique où les développeurs sont contraints de continuer à vérifier leur code, à la recherche de bugs potentiels, pouvant le rendre incompatible avec le projet d'ensemble.

C. Bamboo vs Jenkins

Bamboo

Bamboo d'Atlassian⁶ est un serveur de CI, qui assemble builds, tests et releases automatisées dans un seul enchaînement. Bamboo fournit des intégrations étroites a d'autres outils de développement que Jenkins « en particulier d'autres produits Atlassian » fournissant une visibilité de bout-en-bout dans l'implémentation, la qualité et le statut de tout projet de développement logiciel.

Bamboo supporte les projets basés sur des enchaînements pour gérer builds et tests. Les enchaînements sont organisés par Plans contenant un traitement ou plus, qui incluent une Tâche ou plus. Bamboo supporte l'exécution de tâches en parallèle, comme les tests automatisés.

Jenkins

Jenkins est un outil d'intégration continu Open Source écrit en Java. Jenkins est un fork, crée_ par les développeurs de Hudson après un désaccord avec Oracle. Il supporte les outils SCM tels que

Subversion, Git, Rational ClearCase, etc. Jenkins peut également exécuter des scripts shell et des projets Ant ou Maven. Jenkins dispose en outre de nombreux plugins 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

⁶ Atlassian est une société australienne de logiciels, qui développe des produits pour la gestion de développement et de projets.

APPROCHE METHODOLOGIQUE

applications à grande échelle très rapidement et supporte conception, déploiement et automatisation dans la plupart des projets.

Comparaison Jenkins – Bamboo

	Jenkins	Bamboo
Open Source	Oui	Non
Plugins	Oui Beaucoup de plugins gratuits	Oui Cher
Support	Oui Très large communauté	Oui Peu de communauté
Simple d'utilisation	Non Beaucoup aider par les plugins	Oui
Intégration produits Atlassian	Oui	Oui
Support dépôts	Tout	Tout

Tableau 4-2: Tableau comparatif Jenkins - Bamboo

4.3. Les outils d'analyse de la qualité de code

La revue de code fait partie du processus de développement de logiciels qui consiste à tester le code source pour identifier les bugs à un stade précoce. Un processus de revue de code est généralement mené avant la fusion avec le code de base. Une revue efficace du code permet d'éviter que des bugs et des erreurs ne se glissent dans votre projet en améliorant la qualité du code à un stade précoce du processus de développement du logiciel.

A. Périmètre d'action

L'objectif principal du processus de la revue de code est d'évaluer tout nouveau code afin de détecter les bugs, les erreurs et les normes de qualité axées par l'organisation. Le processus de revue de code ne doit pas se limiter à un simple retour d'information unilatéral. Par conséquent, un avantage intangible du processus de revue de code est l'amélioration collective des compétences de codage de l'équipe.

B. Bénéfices

Le processus de revue de code est essentiel car il ne fait jamais partie du programme scolaire officiel. Vous pouvez apprendre les nuances d'un langage de programmation et de la gestion de projet, mais la revue de code est un processus qui évolue avec l'âge d'une organisation.

La revue de code est essentielle pour les raisons suivantes :

- Vous assurez qu'il n'y a pas de bogue dans le code.
- Minimiser vos chances d'avoir des problèmes.
- Confirmer que le nouveau code respecte les lignes directrices.
- Augmenter l'efficacité du nouveau code.

C. Sonarqube vs Veracode vs Codacy

SonarQube

SonarQube⁷ est une plateforme open source pour l'inspection continue de la qualité de code permettant d'analyser automatiquement le code pour détecter les bugs, les odeurs de code et les failles de sécurité sur plus de 20 langages de programmation⁸. SonarQube offre des

⁷ SonarQube est un logiciel libre de Qualimétrie en continu de code. Il aide à la détection, la classification et la résolution de défaut dans le code source.

⁸ Java (incluant Android), PHP, JavaScript, C/C ++, COBOL, PL/SQL, PL/I, ABAP, VB.NET, VB6, Python, RPG, Flex, Objective-C, Swift, Web et XML. Notez que certains d'entre eux sont commerciaux

APPROCHE METHODOLOGIQUE

rapports sur le code dupliqué, les normes de codage, les tests unitaires, la couverture de code, la complexité du code, les commentaires, les bugs et les failles de sécurité. SonarQube peut enregistrer l'historique des métriques et fournir des graphiques d'évolution. Le plus grand atout de SonarQube est qu'il fournit une analyse et une intégration entièrement automatisées avec Maven, Ant, Gradle, MSBuild et des outils de CI (Atlassian Bamboo, Jenkins, Hudson, etc.). SonarQube s'intègre également aux environnements de développement⁹ à travers les plugins SonarLint et s'intègre aux outils externes tels que LDAP, Active Directory, GitHub, etc. SonarQube est extensible avec l'utilisation de plugins.

Veracode

Veracode aide les entreprises qui innovent par le biais de logiciels à fournir un code sécurisé dans les délais impartis. Contrairement aux solutions sur site qui sont difficiles à adapter et qui se concentrent sur la recherche plutôt que sur la réparation, Veracode comprend une combinaison unique de technologie SaaS et d'expertise à la demande qui permet aux DevSecOps de s'intégrer à votre pipeline, de permettre aux développeurs de corriger les défauts de sécurité et d'adapter votre programme grâce aux meilleures pratiques pour atteindre les résultats souhaités. Veracode couvre tous vos besoins AppSec¹⁰ en une seule solution grâce à une combinaison de cinq types d'analyse disponibles pour 24 langages de programmation, 77 Framework et des types d'application aussi vérifiés que les micro-services, les applications mainframe et mobiles.

Codacy

⁹ Eclipse, Visual Studio et IntelliJ IDEA

¹⁰ AppSec est le processus qui consiste à trouver, corriger et prévenir les vulnérabilités de sécurité au niveau des applications, dans le matériel, les logiciels et les processus de développement. Il comprend des conseils sur les mesures à prendre pour la conception et le développement des applications et tout au long du cycle de vie, y compris après le lancement de l'application.

APPROCHE METHODOLOGIQUE

Codacy est un outil automatisé de révisions du code qui permet aux développeurs d'améliorer la qualité du code et de contrôler la dette technique. Codacy automatise les révisions de code et surveille la qualité du code pour chaque commit et pull request. Il rend compte de l'impact de chaque commit ou pull Request dans les nouveaux numéros concernant les meilleures pratiques de sécurité du style de code et bien d'autres. Elle surveille les changements dans la couverture du code, la duplication du code et la complexité du code. Il permet aux développeurs de gagner du temps dans les révisions de code et de s'attaquer efficacement à la dette technique.

Comparaison Sonarqube, Veracode, Codacy

Caractéristiques	Sonarqube	Veracode	Codacy
Open Source	Oui	Oui	Oui
# dépôts privés	Illimité	Illimité	Illimité
# contributeurs	3	Illimité	Illimité
CI	Oui	Oui	Oui
CD	Oui	Oui	Oui

Tableau 4-3: Tableau comparatif Sonarqube - Veracode – Codacy

4.4. Les outils de gestion de configuration

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.

A. Périmètre d'action

La gestion de configuration garantit l'homogénéité d'une infrastructure d'un grand nombre de façons, en fonction de l'outil choisi. La possibilité de gérer votre infrastructure en tant que code (infrastructure as code) est un autre aspect de la gestion de configuration, et pour un grand nombre d'entreprises, il s'agit de la solution la plus simple. Toutefois, des méthodes de gestion de configuration plus traditionnelles sont disponibles, et toutes les configurations sont possibles dès lors que les composants système sont modifiables en fonction des prérequis et d'une manière homogène. Cette souplesse vous permet de revenir rapidement à un état antérieur, les implémentations n'ont pas besoin d'être répétées, et vous ne ferez pas l'erreur de remplacer le mauvais composant.

B. Bénéfices

La fonction première des outils de gestion de configuration est d'éviter les incidents et de réduire les interruptions de service. De plus, une visibilité et un suivi améliorés confèrent aux services informatiques davantage de stabilité et de contrôle. Le retour sur investissement peut s'avérer très élevé, car les outils de gestion de configuration ne se contentent pas de réduire les tâches superflues, les duplications et d'éliminer toute confusion ; ils améliorent également la productivité des équipes en les affranchissant des tâches de maintenance basiques, ce qui réduit le temps passé à gérer des crashes système et le coût de la gestion d'infrastructure. Les avantages

ne s'arrêtent pas là, puisque les outils réduisent considérablement les risques et améliorent la sécurité.

Enfin, une entreprise qui implémente des outils de gestion de configuration peut optimiser le nombre de serveurs et éviter tout dépassement aboutissant à des interruptions de service, majeures ou mineures.

C. Puppet vs Chef vs Ansible

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.

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.

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, versionnable, lisible et testable, quelle que soit la plateforme où il s'exécute ou la taille du réseau.

APPROCHE METHODOLOGIQUE

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.

Ansible

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

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, _able, et convivial, contenant un minimum de dépendances. De plus, son architecture sans agents garantit sa sécurité et réduit les frais de réseau.

Comparaison Puppet, Chef, Ansible

APPROCHE METHODOLOGIQUE

CARACTERISTIQUES	CHEF	PUPPET	ANSIBLE
DISPONIBILITE (EN CAS DE PANNE)	Backup Server	Alternative Master	Instance secondaire
LANGAGE DE CONFIGURATION	Ruby DSL	Ruby, Puppet DSL, Embedded Ruby (ERB), DSL	Python, YAML
ARCHITECTURE	Master-Agent	Master-Agent	Agentless
PROCESSUS D'INSTALLATION	Longue et Complexe	Longue et Complexe	Facile
GESTION DE CONFIGURATION	Pull	Pull	Push et Pull
SCALABILITE	Haute	Haute	Très Haute
INTEROPERABILITE	Serveur CHEF fonctionne uniquement sur Linux/Unix	Master CHEF fonctionne uniquement sur Linux/Unix	Serveur Ansible fonctionne sur Linux/Unix
CAPACITE	<ul style="list-style-type: none"> • CD avec un workflow automatisé • Gestion de la conformité et de la sécurité • Automatisation des infrastructures 	<ul style="list-style-type: none"> • Orchestration • Approvisionnement automatique • Gestion du code et des nœuds • Automatisation des configurations • Visualisation et rapports • Grande transparence • Contrôle d'accès basé sur les rôles 	<ul style="list-style-type: none"> • Orchestration simple • Approvisionnement simplifié • CD avec un workflow automatisé • Déploiement d'applications • Intégration de la sécurité et de la conformité dans l'automatisation

Tableau 4-4: Tableau comparatif Puppet - Chef - Ansible

4.5. Les outils de messagerie et de collaboration

Travailler et créer ensemble est crucial pour chaque industrie et organisation où les équipes sont réparties dans le monde entier. Cela devient d'autant plus important pour les organisations informatiques où le développement d'applications exige une collaboration cohérente et efficace entre les équipes qui s'occupent du développement, des tests, de l'analyse et de la production.

A. Pourquoi la collaboration est-elle cruciale pour DevOps ?

La collaboration entre les équipes DevOps d'une organisation peut servir de pont entre la technologie et les processus. Une collaboration efficace peut permettre aux membres de l'équipe de se concentrer sur les activités de développement des applications de base et d'optimiser les frais généraux opérationnels.

De plus en plus d'organisations se rendent compte de l'importance des DevOps. Selon le rapport 2018 Global Developer Report de GitLab [Git], 65% des entreprises estiment que le workflow DevOps leur permet de gagner du temps pendant le processus de développement. Une majorité de développeurs (81%) conviennent qu'il est important de travailler dans un environnement collaboratif. Voici quelques-uns des plus gros problèmes qui peuvent survenir si les équipes DevOps ne collaborent pas efficacement :

- Redondance du travail due aux silos de communication qui empêchent le partage des idées.
- Les transferts improductifs entre les équipes de développement et d'exploitation de logiciels.
- Confusion entre les développeurs quant à savoir qui est responsable de quelles tâches.
- Impact négatif sur la stabilité des produits et les cycles de lancement.
- Des inefficacités dans les opérations qui entraînent des retards dans les livraisons.
- Une quantité massive de courriers électroniques, le mécontentement des utilisateurs finaux et la perte de revenus.

Il ne fait aucun doute que les équipes DevOps doivent mieux rationaliser l'ensemble du cycle de vie du développement de logiciels grâce à une collaboration efficace.

B. Que signifie réellement la collaboration pour DevOps ?

Le mot "collaboration" est utilisé à plusieurs reprises dans le cadre de DevOps, mais que signifie-t-il réellement dans le contexte de DevOps ?

Les experts estiment que chaque fois qu'une organisation annonce qu'elle va mettre en œuvre DevOps, la première chose qui devrait venir à l'esprit de chacun est la collaboration. C'est l'intimité entre DevOps et la collaboration.

Chaque organisation devrait s'y intéresser dès le début du projet. Lorsque le développeur et les équipes IT travaillent ensemble, cela permet non seulement de gagner du temps, mais aussi de trouver et de résoudre rapidement les problèmes. Lorsqu'il y a un manque de collaboration, le développeur peut dire que c'est le problème de l'équipe IT, et l'équipes IT peut penser que c'est le problème du développeur.

Ainsi, grâce à une bonne collaboration, les équipes de développement et d'exploitation peuvent se mettre ensemble sur un projet, surveiller et enregistrer les performances, et résoudre les problèmes dès que possible, en gardant à l'esprit l'orientation client.

C. Slack et Trello

Slack

Slack, la plate-forme de collaboration quasi universelle développée par la société de logiciels américaine Slack Technologies, fournit un ensemble d'outils tels que la messagerie directe, les appels audio/vidéo et le partage de documents pour aider à améliorer la collaboration au sein de l'équipe.

Outre les chats de groupe et individuels, Slack offre des fonctionnalités intéressantes comme les réponses en ligne, la polyvalence mobile et l'intégration avec un grand nombre de services tiers comme GitHub, Dropbox, Box, Google Drive, Heroku, IBM Bluemix, Zendesk, Trello, Crashlytics, Runscope et Zapier. Slack est un produit freemium. Cela signifie que vous pouvez

utiliser plusieurs de ses fonctions gratuitement, tandis que certaines fonctions améliorées, comme la prise en charge d'un nombre illimité d'utilisateurs et l'intégration d'applications, peuvent être utilisées à un prix élevé. Pour cette raison, Slack a gagné une bonne popularité auprès des équipes de toutes tailles, qu'il s'agisse d'entreprises, de petites organisations ou de particuliers.

Trello

Trello est un outil de collaboration basé sur des listes de tâches qui permet aux équipes DevOps d'utiliser des listes, des groupes et des cartes partagés de manière simple. Il offre une vue unique de l'état d'avancement du projet, y compris les activités en cours, déjà terminées et encore à démarrer, sous la forme d'un tableau blanc unique et facile à gérer. Chaque tâche peut être accompagnée de fichiers ou d'images, et peut également être directement reliée à d'autres sources de données comme BitBucket ou Salesforce, avec la possibilité de commenter et de collaborer avec tous les membres de l'équipe. Les tableaux Trello sont également équipés de "power-ups" ¹¹ qui permettent à l'équipe d'utiliser le dépôt de code, le suivi des bugs et le suivi du temps pour les projets

Trello. Il est disponible en deux options de licence, une version gratuite pour les particuliers, et la version payante par abonnement pour les entreprises, ayant des fonctions d'équipes avancées et des intégrations d'applications.

4.6. Les outils de suivi de projet/d'anomalies

Le suivi d'un projet - délai du projet, gestion des blocages potentiels, modification de la portée initiale du projet, et rapport d'anomalies - est un casse-tête bien connu pour les entreprises

¹¹ Capacité d'intégration et de développement d'applications

de toute taille. Les outils de suivi de projets et d'anomalies sont conçus pour simplifier ces processus.

Ce sont des plateformes dédiées qui permettent aux équipes de suivre leurs projets, du démarrage à la livraison, de la détection d'anomalie à sa résolution. Les outils de suivi de projets aident les équipes à rester concentrées, grâce à des échéanciers visuels sur les progrès, modifiables en quelques secondes. Ils structurent le projet et garantissent sa livraison en éliminant les dépendances et en unifiant les progrès du projet en un seul endroit : plus besoin de recourir aux emails et aux tableurs Excel.

A. Périmètre d'action

Fondamentalement, les outils de suivi de projet sont conçus pour rationaliser les tâches qui constituent le quotidien de la vie professionnelle. Ils permettent aux utilisateurs de devenir plus productifs, à moindre effort en optimisant leur emploi du temps. Échéanciers, Scrum personnalisable, tableaux de bord Kanban sont autant de fonctionnalités permettant aux équipes de décomposer leurs livrables en petites tâches, simples à gérer et itératives, qui apportent progressivement de la valeur ajoutée au produit. De nombreux outils comprennent des backlogs partagés, permettant aux individus de visualiser la portée du projet : ce dernier reste ainsi agile, mais maîtrisé.

B. Bénéfices

Les outils de suivi de projets et d'anomalies offrent un grand nombre de bénéfices. Ils confèrent un niveau de rationalisation tel, que les dépendances logicielles deviennent superflues. Ils garantissent par ailleurs la centralisation du suivi de projet. Un point unique de référence pour le statut du projet, modifiable en temps réel, confère une grande visibilité sur les progrès de chaque équipe.

Les outils, certains étant Open Source, offrent différents niveaux de fonctionnalités. Vous pouvez ainsi choisir la solution qui convient le mieux à votre organisation.

C. Jira

JIRA Software, d'Atlassian, est un outil de gestion de projet pour équipes agiles. Ces dernières peuvent se concentrer sur la livraison itérative et progressive de projets à haute valeur ajoutée avec des tableaux Scrum personnalisables. Les tableaux Kanban fournissent une visibilité totale aux équipes sur les projets à venir, pour qu'elles puissent livrer le maximum de résultats en un cycle de temps minimal. Les équipes ont accès à plus d'une douzaine de rapports prêts à l'emploi en temps réel sur leur performance à chaque sprint. Un module de gestion agile permet aux utilisateurs de planifier et anticiper des feuilles de route prévisibles, et prendre des décisions en connaissance de cause.

4.7. Les conteneurs

Bien que les conteneurs existent depuis quelque temps déjà, leur popularité a récemment explosé. Cette technologie permet essentiellement d'exécuter des applications dans des environnements virtuels en stockant tous les fichiers, bibliothèques, et dépendances au sein d'un seul package.

Ce mode de fonctionnement découle principalement de l'adaptation modernes de la technologie par des entreprises telles que Docker et Kubernetes : leurs produits innovants permettent de gérer individuellement des traitements distincts, que ce soit sous Windows, Linux, ou Unix. Notons toutefois que, même si l'architecture de Google, Facebook, ou autres entreprises similaires est conçue sur les conteneurs, l'approche est différente dans les environnements des autres entreprises. Ce fonctionnement nécessitera de revoir l'architecture, de réécrire les applications, et de nouvelles équipes pour mettre en œuvre leur développement.

A. Périmètre d'action

Les conteneurs fonctionnent en se connectant directement au système d'exploitation. Nul besoin d'une nouvelle VM pour exécuter différentes applications sur le même système d'exploitation : le kernel de l'O/S peut en effet être partagé entre différents services et

applications. De plus, ils ne nécessitent pas la création d'une nouvelle machine virtuelle à chaque fois que vous avez besoin d'une nouvelle instance de l'application. En pratique, le logiciel peut être déplacé d'un environnement à un autre et s'exécuter très efficacement.

B. Bénéfices

La portabilité et la souplesse de la conteneurisation offrent de nombreux bénéfices à ses utilisateurs. Il s'agit de la méthode la plus intuitive de développement d'applications basées sur le Cloud ; et les conteneurs sont très légers, comparés aux machines virtuelles traditionnelles : l'exécution d'approximativement 17 conteneurs nécessite le même hardware qu'une seule machine virtuelle. Ils offrent par ailleurs une élasticité native, adaptant leur puissance de calcul à vos besoins. En outre, les conteneurs apportent une couche supplémentaire de simplicité. Il suffit d'une seule ligne de commande pour créer des requêtes, comparé au paramétrage, à la configuration et à l'installation de machines virtuelles plus traditionnelles.

C. Docker vs rkt from CoreOS vs LXC

Docker

Docker est un projet Open Source qui automatise le déploiement des applications dans les conteneurs logiciels. Les conteneurs Docker rassemblent des binaires applicatifs et des éléments de configuration au sein d'un système de fichier isolé et d'un espace mémoire qui inclut également tout ce dont une application a besoin pour s'exécuter : code, runtime, outils système et bibliothèques système.

Docker utilise des fonctionnalités d'isolation de ressources de Linux et Windows et fournit une couche supplémentaire d'abstraction et d'automatisation de virtualisation au niveau du système d'exploitation. Docker garantit que ses images de conteneur s'exécuteront toujours de la même façon, quel que soit l'environnement hébergé. Les développeurs utilisent les dépendances app du package Docker au sein des conteneurs pour assurer portabilité et prédictibilité pendant développement, test et déploiement.

rkt from CoreOS

Le plus grand concurrent de Docker sur le marché de la virtualisation basée sur des conteneurs est l'environnement d'exécution, rkt (prononcer "rocket" ou "rock-it") du distributeur de Linux, CoreOS.

Contrairement à Docker, le concurrent visait à impressionner avec ses caractéristiques de sécurité supplémentaires. Celles-ci comprennent, en plus de l'isolation des conteneurs sur une base

KVM, la prise en charge de l'extension de noyau SELinux, ainsi que la validation de la signature pour les images de la spécification de conteneur auto-développée App Container (appc). Celle-ci décrit le format d'image App Container Image (ACI), l'environnement d'exécution, un mécanisme de découverte d'images et la possibilité de regrouper les images dans des applications multi-conteneurs, connues sous le nom d'App Container Pods.

rkt prend en charge d'autres formats que ses propres images de conteneur. L'environnement d'exécution est compatible avec les images Docker et, avec l'outil open source Quay, tout format de conteneur peut être converti en ACI.

rkt n'est pas limité aux fonctions du noyau Linux telles que les cgroupes et les espaces de noms lors de la virtualisation d'applications. Avec l'environnement d'exécution de CoreOS, les conteneurs basés sur KVM (machine virtuelle basée sur le noyau, par exemple LKVM ou QEMU) et la technologie Intel Clear Container peuvent également être lancés en tant que machines virtuelles entièrement fermées.

LXC

L'alternative au Docker, LXC, est un ensemble d'outils, de modèles, de bibliothèques et de liens linguistiques qui, ensemble, représentent une interface en espace utilisateur pour les

fonctions de conteneur natif du noyau Linux. Pour les utilisateurs de Linux, LXC offre un moyen pratique de créer et de gérer des conteneurs d'applications et de systèmes.

Pour protéger les processus les uns des autres, LXC utilise les techniques d'isolation suivantes :

- IPC, UTS, Mount, PID, réseau et espaces de noms d'utilisateurs
- Cgroupes
- Profil AppArmor et SELinux
- Règles Seccomp
- Chroots
- Capacités du noyau

Le but du projet LXC est de créer un environnement de conteneur de logiciels qui soit aussi similaire que possible à une installation Linux standard. LXC est développé en parallèle avec les projets open source LXD, LXCFS et CGManager dans le cadre du projet LinuxContainers.org.

LXC a été conçu pour faire fonctionner différents conteneurs de système (conteneurs de système complet) sur un système hôte commun. Un conteneur Linux lance généralement une distribution complète à partir d'une image de système d'exploitation dans un environnement virtuel. Les utilisateurs interagissent avec lui de la même manière qu'avec une machine virtuelle.

Une différence majeure entre les deux technologies de virtualisation est que les conteneurs Linux peuvent contenir un nombre quelconque de processus, alors qu'un seul processus est exécuté dans les conteneurs Docker, de sorte que les applications complexes de Docker sont généralement constituées de plusieurs conteneurs. Un déploiement efficace d'applications multi-conteneurs de ce type nécessite des outils supplémentaires.

APPROCHE METHODOLOGIQUE

En outre, les conteneurs Docker et les conteneurs Linux diffèrent en termes de portabilité. Si un utilisateur développe un logiciel basé sur LXC sur un système de test local, il ne peut pas simplement supposer que le conteneur fonctionnera parfaitement sur d'autres systèmes (par exemple un système productif). La plateforme Docker, cependant, extrait beaucoup plus fortement les applications de la qualité du système sous-jacent. Cela permet au même conteneur Docker d'être exécuté sur n'importe quel hôte Docker (un système sur lequel le moteur Docker a été installé), indépendamment du système d'exploitation et de la configuration matérielle.

LXC est également livré sans processus de démon central. Au lieu de cela, le logiciel du conteneur s'intègre dans les systèmes d'init tels que systemd et upstart pour démarrer et gérer les conteneurs.

Comparaison Docker, rkt, LXC

Caractéristiques	Docker	rkt	LXC
Open Source	Oui	Oui	Oui
# dépôts privés	Illimité	Illimité	Illimité
# contributeurs	3	Illimité	Illimité
CI	Oui	Oui	Oui
CD	Oui	Oui	Oui

Tableau 4-5: Tableau comparatif Docker - rkt - LXC

4.8. Les outils d'orchestration de conteneurs

L'orchestration des conteneurs permet d'automatiser le déploiement, la gestion, la mise à l'échelle et la mise en réseau des conteneurs. Les entreprises qui ont besoin de déployer et de gérer des centaines ou des milliers de conteneurs Linux et d'hôtes peuvent tirer parti de l'orchestration des conteneurs.

Cette technologie est compatible avec tous les environnements qui exécutent des conteneurs. Elle permet de déployer la même application dans différents environnements sans modifier sa conception. De plus, les microservices stockés dans les conteneurs simplifient l'orchestration des services, notamment les services de stockage, de réseau et de sécurité.

Avec les conteneurs, les applications basées sur des microservices disposent d'une unité de déploiement et d'un environnement d'exécution parfaitement adaptés. Les conteneurs permettent d'exécuter plusieurs parties d'une application dans des microservices, indépendamment les uns des autres, sur le même matériel, avec un niveau de contrôle bien plus élevé sur leurs éléments et cycles de vie.

La gestion du cycle de vie des conteneurs avec l'orchestration aide également les équipes DevOps qui l'intègrent dans les workflows CI/CD. Avec les interfaces de programmation d'application (API) et les pratiques DevOps, les microservices conteneurisés constituent la base des applications cloud-native.

A. A quoi sert l'orchestration des conteneurs ?

Nous pouvons utiliser l'orchestration des conteneurs pour automatiser et gérer les tâches suivantes :

- Approvisionnement et déploiement
- Configuration et planification

- Allocation des ressources
- Mise à disposition des conteneurs
- Mise à l'échelle ou suppression de conteneurs en fonction des charges de travail dans l'infrastructure
- Equilibrage de la charge et routage du trafic
- Surveillance de l'intégrité des conteneurs
- Configuration des applications en fonction du conteneur sur lequel elles vont s'exécuter
- Sécurisation des interactions entre les conteneurs

B. Kubernetes vs OpenShift vs Nomad

Kubernetes

Kubernetes (communément appelé « K8s¹² ») est un système open source qui vise à fournir une « plate-forme permettant d'automatiser le déploiement, la montée en charge et la mise en œuvre de conteneurs d'application sur des clusters de serveurs ». Il fonctionne avec toute une série de technologies de conteneurisation, et est souvent utilisé avec Docker. Il a été conçu à l'origine par Google, puis offert à la Cloud Native Computing Foundation.

Kubernetes définit un jeu d'outils ("primitives") qui, ensemble, fournissent des mécanismes pour déployer, maintenir et mettre à l'échelle des applications. Ces éléments qui composent Kubernetes sont conçus pour être combinés et extensibles et donc permettre de supporter une grande variété de charge de travail. Cette extensibilité est fournie en grande partie par l'API de Kubernetes, qui est utilisée par les composants internes aussi bien que par les extensions et les conteneurs tournant sur Kubernetes.

OpenShift

¹² K8s est une abréviation obtenue en remplaçant les 8 lettres "ubernete" par "8"

La solution Red Hat OpenShift est une plateforme Kubernetes de pointe : sa base cohérente et axée sur la sécurité permet de distribuer des applications n'importe où tandis que ses workflows de développement rationalisés accélèrent la mise sur le marché. Avec la solution Red Hat OpenShift, les entreprises qui innovent restent concentrées sur l'essentiel, à savoir, garder une longueur d'avance sur la concurrence et s'appliquer à dépasser les attentes de leurs clients.

La solution Red Hat OpenShift offre tout ce dont vous avez besoin pour le cloud hybride et les conteneurs d'entreprise, mais également pour le développement et les déploiements Kubernetes.

La plateforme inclut un système d'exploitation Linux d'entreprise, un environnement d'exécution pour les conteneurs ainsi que des solutions de mise en réseau, de surveillance, de registre de conteneur, d'authentification et d'autorisation. Tous ces composants sont testés ensemble pour garantir une exploitation cohérente sur une plateforme Kubernetes complète qui englobe tous les types de clouds.

Nomad

Nomad est un gestionnaire de cluster et un planificateur conçu pour les microservices et les charges de travail par lots. Nomad est distribué, hautement disponible, et s'adapte à des milliers de nœuds couvrant plusieurs centres de données et régions.

Nomad fournit un flux de travail commun pour déployer des applications dans une infrastructure. Les développeurs utilisent une spécification de travail déclarative pour définir comment une application doit être déployée et les ressources dont elle a besoin (CPU, mémoire, disque). Nomad accepte ces emplois et trouve les ressources disponibles pour les exécuter. L'algorithme de planification garantit que toutes les contraintes sont satisfaites et qu'il empaquette autant d'applications que possible sur un hôte afin d'optimiser l'utilisation des

APPROCHE METHODOLOGIQUE

ressources. En outre, Nomad prend en charge les applications virtualisées, conteneurisées ou autonomes s'exécutant sur tous les principaux systèmes d'exploitation, ce qui lui donne la souplesse nécessaire pour prendre en charge un large éventail de charges de travail.

Comparaison Kubernetes, OpenShift, Nomad

Caractéristiques	Kubernetes	OpenShift	Nomad
Open Source	Oui	Oui	Oui
# dépôts privés	Illimité	Illimité	Illimité
# contributeurs	3	Illimité	Illimité
CI	Oui	Oui	Oui
CD	Oui	Oui	Oui

Tableau 4-6: Tableau comparatif Kubernetes - OpenShift - Nomad

5. MISE EN ŒUVRE

5.1. Architecture Continuous Deployment

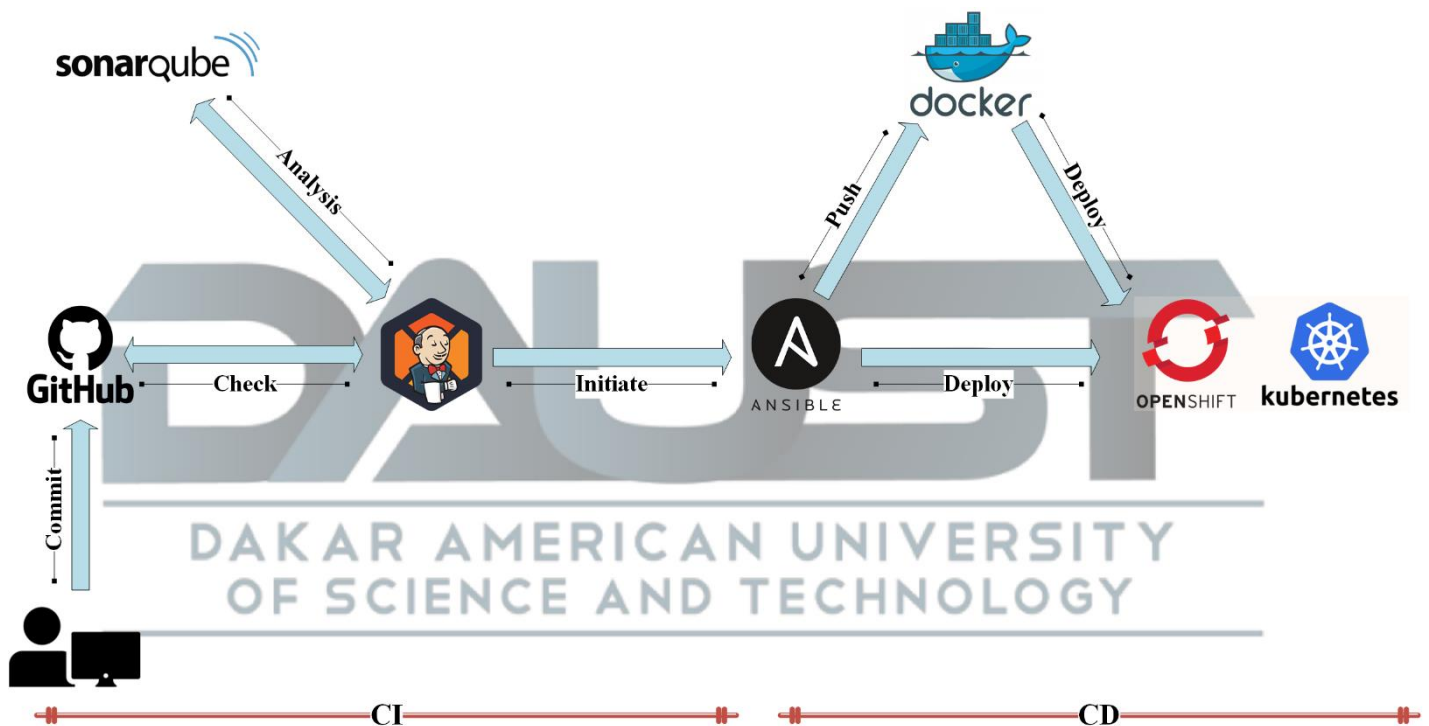


Figure 5-1: Architecture Continuous Deployment

Dans cette architecture, la méthode de gestion de projet utilisé est SCRUM. Dans JIRA après avoir défini le Product Backlog, le sprint peut démarrer.

Chaque développeur s'assigne une ou plusieurs tâche(s) et commence à implémenter la/les fonctionnalité(s). Une fois que la fonctionnalité finie, il fait un commit dans le SCM. Jenkins détecte automatiquement que le repository a été modifié et lance le pipeline. Le code nouvellement mis à jour est récupéré depuis le SCM, le code est buildé et les tests unitaires sont réalisés. Une fois les tests unitaires passés, Jenkins l'envoie à SonarQube pour l'analyse statique du code. Après analyse, SonarQube envoie un rapport à Jenkins lui notifiant si toutes les règles

mises en place à savoir une bonne documentation, un taux de couverture et un niveau de sécurité acceptable, entre autres, sont respectées. Si le rapport de l'analyse est validé le code est soit :

- Packagé en WAR/JAR dans le cas d'un code JAVA
- Buildé en une image Docker.

Ou les deux à la fois. Pour la prochaine étape ; Ansible se charge du déploiement de l'image Docker dans un container Docker distant ou le déploiement du War/Jar dans serveur distant.

5.2. Fonctionnement des différents éléments de l'architecture

A. Jenkins, pourquoi utiliser les pipelines ?

Si les emplois "libres" standard de Jenkins favorisent une intégration simple et continue en permettant de définir des tâches séquentielles dans le cycle de vie d'une application, ils ne créent pas d'enregistrement persistant de l'exécution, ne permettent pas à un script de traiter toutes les étapes d'un flux de travail complexe, ni ne confèrent les autres avantages des pipelines.

Contrairement aux emplois de style libre, les pipelines permettent de définir l'ensemble du cycle de vie des applications. Les pipelines aident Jenkins à prendre en charge la livraison continue (CD). Le plugin Pipeline a été conçu en tenant compte des exigences d'un flux de travail flexible, extensible et basé sur des scripts pour les CD.

En conséquence, la fonctionnalité du pipeline est :

- Durable : Les pipelines peuvent survivre aux redémarrages prévus et imprévus du serveur Jenkins.
- Pausable : Les pipelines peuvent éventuellement s'arrêter et attendre une intervention humaine ou une approbation avant de terminer les travaux pour lesquels ils ont été construits.

- Polyvalent : les pipelines prennent en charge les besoins complexes des CD du monde réel, notamment la possibilité de bifurquer(fork) ou de joindre, de faire des boucles et de travailler en parallèle les uns avec les autres
- Efficace : Les pipelines peuvent redémarrer à partir de n'importe lequel des nombreux points de contrôle sauvegardés.
- Extensible : Le plugin Pipeline supporte des extensions personnalisées de son DSL (Domain Scripting Language) et de multiples options d'intégration avec d'autres plugins.

Définition du pipeline

Les pipelines sont des emplois Jenkins activés par le plugin Pipeline (anciennement appelé "workflow") et construits avec de simples scripts de texte qui utilisent un DSL (Domain Specific Language) basé sur le langage de programmation Groovy. Les pipelines exploitent la puissance de plusieurs étapes pour exécuter des tâches simples et complexes selon des paramètres que vous établissez. Une fois créés, les pipelines peuvent construire du code et orchestrer le travail nécessaire pour mener les applications de la validation à la livraison.

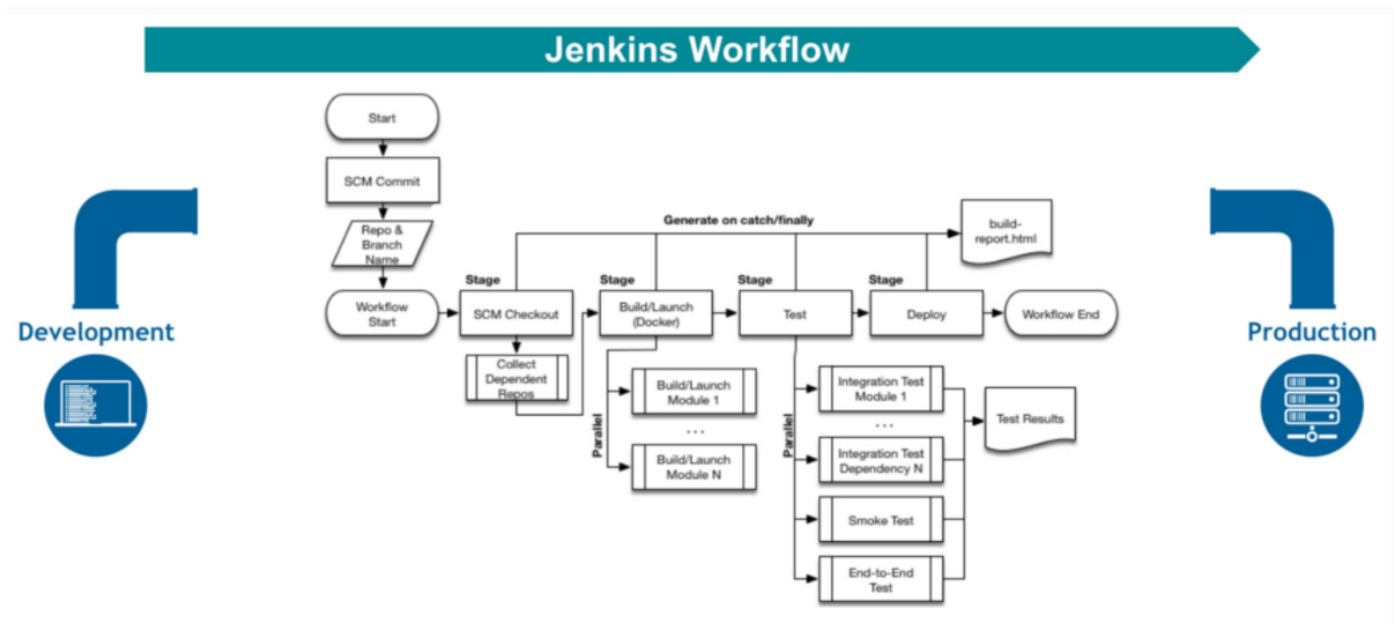


Figure 5-2: Exemple scénario de livraison continue avec les pipelines

Vocabulaire des pipelines

Les termes de pipeline tels que "step", "node¹³" et "stage" sont un sous-ensemble du vocabulaire utilisé pour Jenkins en général.

Step

Un step (souvent appelée "build step") est une tâche unique qui fait partie d'une séquence. Les steps indiquent à Jenkins ce qu'il doit faire.

Node

Dans les contextes de codage en pipeline, contrairement à ce que fait généralement Jenkins, un "nœud" est une étape qui fait deux choses, généralement en faisant appel à l'aide des exécuteurs disponibles sur les agents :

¹³ Dans le cadre de Jenkins en général, un "node" désigne tout ordinateur faisant partie de votre installation

Jenkins, que cet ordinateur soit utilisé comme contrôleur ou comme agent

- Planifie l'exécution des étapes qu'il contient en les ajoutant à la file d'attente de construction Jenkins (de sorte que dès qu'un créneau d'exécution est libre sur un nœud, les étapes appropriées s'exécutent).
- Crée un espace de travail, c'est-à-dire un répertoire de fichiers spécifique à un travail particulier, où un traitement gourmand en ressources peut avoir lieu sans nuire aux performances de votre pipeline. Les espaces de travail durent pendant toute la durée des tâches qui leur sont assignées.

Stage

Un stage est une étape qui fait appel aux API prises en charge. La syntaxe du pipeline est composée d'étapes. Chaque étape peut comporter une ou plusieurs étapes de construction. La connaissance des termes de Jenkins tels que "contrôleur", "agent" et "exécuteur" aide également à comprendre le fonctionnement des pipelines. Ces termes ne sont pas spécifiques aux pipelines :

- Un "contrôleur" est l'installation de base de Jenkins sur un ordinateur ; il gère les tâches de votre système de construction. Les scripts de pipeline sont analysés sur les contrôleurs, et les étapes enveloppées dans des blocs de nœuds sont exécutées sur les exécuteurs disponibles.
- Un "agent" (anciennement "esclave") est un ordinateur configuré pour décharger des projets particuliers du contrôleur. Votre configuration détermine le nombre et l'étendue des opérations qu'un agent peut effectuer. Les opérations sont effectuées par des exécuteurs.
- Un "exécuteur" est une ressource informatique permettant de compiler du code. Il peut fonctionner sur les machines du contrôleur ou de l'agent, soit par lui-même, soit en parallèle avec d'autres exécuteurs. Jenkins attribue un fil `java.lang` à chaque exécuteur.

B. Docker, qu'est-ce que c'est ?

Le terme « Docker » désigne plusieurs choses : le projet d'une communauté Open Source, les outils issus de ce projet Open Source, l'entreprise Docker Inc. qui constitue le principal soutien de ce projet, ainsi que les outils que l'entreprise prend officiellement en charge. Des technologies et une entreprise qui partagent le même nom, cela peut prêter à confusion. Voici donc une rapide explication de cet état de fait.

- Le logiciel « Docker » est une technologie de conteneurisation qui permet la création et l'utilisation de conteneurs Linux.
- La communauté Open Source Docker travaille à l'amélioration de cette technologie disponible gratuitement pour tout le monde.
- L'entreprise Docker Inc. s'appuie sur le travail de la communauté Docker, sécurise sa technologie et partage ses avancées avec tous les utilisateurs. Elle prend ensuite en charge les technologies améliorées et sécurisées pour ses clients professionnels.

Avec la technologie Docker, vous pouvez traiter les conteneurs comme des machines virtuelles très légères et modulaires. En outre, ces conteneurs vous offrent une grande flexibilité : vous pouvez les créer, déployer, copier et déplacer d'un environnement à un autre, ce qui vous permet d'optimiser vos applications pour le cloud.

Comment fonctionne la technologie Docker ?

La technologie Docker utilise le noyau Linux et des fonctions de ce noyau, telles que les groupes de contrôle cgroups et les espaces de noms, pour séparer les processus afin qu'ils puissent s'exécuter de façon indépendante. Cette indépendance reflète l'objectif des conteneurs : exécuter plusieurs processus et applications séparément les uns des autres afin d'optimiser l'utilisation de votre infrastructure tout en bénéficiant du même niveau de sécurité que celui des systèmes distincts. Les outils de conteneurs, y compris Docker, sont associés à un modèle de déploiement basé sur une image. Il est ainsi plus simple de partager une application ou un

ensemble de services, avec toutes leurs dépendances, entre plusieurs environnements. Docker permet aussi d'automatiser le déploiement des applications (ou d'ensembles de processus combinés qui forment une application) au sein d'un environnement de conteneurs.

Ces outils conçus sur des conteneurs Linux (d'où leur convivialité et leur singularité) offrent aux utilisateurs un accès sans précédent aux applications, la capacité d'accélérer le déploiement, ainsi qu'un contrôle des versions et de l'attribution des versions.

La technologie Docker est-elle la même que celle des conteneurs Linux traditionnels ?

Non. A l'origine, la technologie Docker a _été créée sur la base de la technologie LXC, que la plupart des utilisateurs associent aux conteneurs Linux « traditionnels », mais elle s'est depuis émancipée. LXC était un outil de virtualisation léger très utile, mais il n'offrait pas une expérience à la hauteur pour les utilisateurs ou les développeurs. La technologie Docker permet non seulement d'exécuter des conteneurs, mais aussi de simplifier leur conception et leur fabrication, l'envoi d'images, le contrôle des versions d'image, etc.

Les conteneurs Linux traditionnels utilisent un système init capable de gérer plusieurs processus.

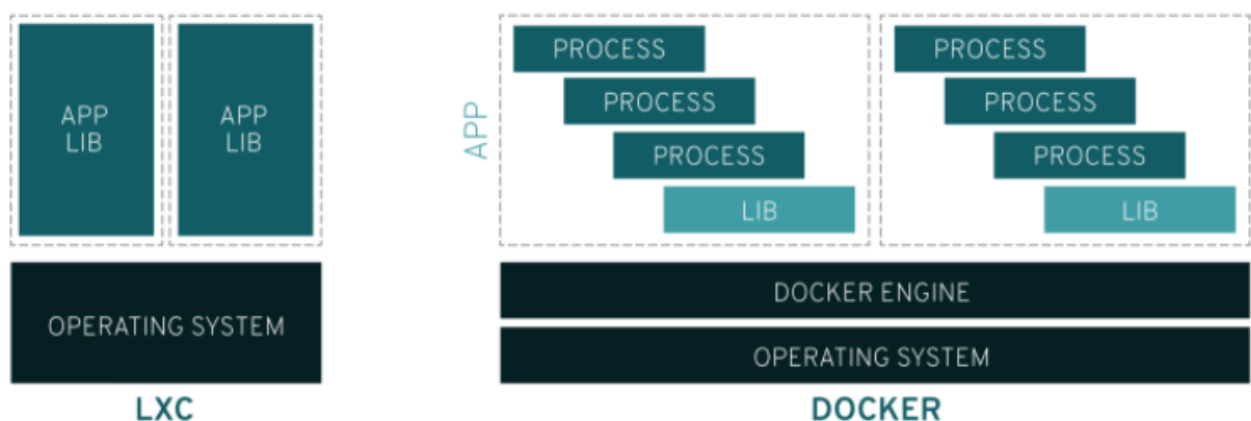


Figure 5-3: Conteneurs traditionnels Linux vs Docker

Ainsi, des applications entières peuvent s'exécuter comme un bloc. La technologie Docker encourage la décomposition des applications en processus distincts et fournit les outils nécessaires pour y parvenir. Cette approche granulaire présente bien des avantages.

C. Le déploiement d'applications avec ansible ?

Ansible est le moyen le plus simple de déployer nos applications. Il nous donne la possibilité de déployer des applications multiniveaux de manière fiable et cohérente, le tout à partir d'un cadre commun. Vous pouvez configurer les services nécessaires ainsi que les artefacts d'application push à partir d'un système commun.

Plutôt que d'écrire du code personnalisé pour automatiser vos systèmes, votre équipe rédige des descriptions de tâches simples que même le membre le plus récent de l'équipe peut comprendre lors de la première lecture, ce qui permet non seulement d'économiser des coûts initiaux, mais également de réagir plus facilement aux changements au fil du temps.

PUISSANCE DES PLAYBOOKS

REPETABLE ET FIABLE

Ansible vous permet d'écrire des « Playbooks » qui sont des descriptions de l'état souhaité de vos systèmes, qui sont généralement conservés sous contrôle de code source. Ansible fait alors le travail acharné pour amener vos systèmes à cet état, quel que soit l'état dans lequel ils se trouvent actuellement. Les playbooks rendent vos installations, mises à niveau et gestion quotidienne répétables et fiables.

SIMPLE A ECRIRE ET A MAINTENIR

Les playbooks sont simples à écrire et à maintenir. La plupart des utilisateurs deviennent productifs avec Ansible après seulement quelques heures. Ansible utilise les mêmes outils que vous utilisez probablement déjà quotidiennement et les playbooks sont écrits dans un langage naturel, ils sont donc très faciles à faire évoluer et à modifier.

AUCUN AGENT = PLUS DE SECURITE, PLUS DE PERFORMANCES, MOINS D'EFFORT

Grâce à sa conception sans agent, Ansible peut être introduit dans votre environnement sans aucun démarrage de systèmes distants ni ouverture de ports supplémentaires. Non seulement cela élimine la « gestion de la gestion », mais l'utilisation des ressources du système est également considérablement améliorée.

Batteries incluses

Tirez parti d'une boîte à outils géante. Livré avec plus de 1 300 modules dans la distribution principale, Ansible fournit une vaste bibliothèque de blocs de construction pour la gestion de toutes sortes de tâches informatiques et de logiciels de réseau. Avec Ansible Galaxy, il y a de fortes chances qu'il existe des rôles contribués par la communauté qui peuvent vous aider à démarrer encore plus rapidement.

Aucun temps d'arrêt

Comme indiqué dans le diagramme ci-dessus, Ansible peut orchestrer les mises à jour progressives sans temps d'arrêt de manière simple, ce qui vous permet de mettre à jour vos applications en production sans que les utilisateurs ne le remarquent.

Super flexible

Le téléchargement d'artefacts à partir de serveurs et la configuration du système d'exploitation ne sont que les bases. Parlez aux API REST, mettez à jour un serveur de chat d'équipe avec un avertissement ou envoyez un e-mail - Ansible peut gérer toutes sortes de flux de travail.

Prêt pour le cloud

Les modules inclus gèrent non seulement le système informatique local, mais peuvent interagir avec les services cloud, notamment Amazon AWS, Microsoft Azure, etc. Et comme

toutes les API cloud vous permettent d'injecter de manière simple des clés SSH, vous pouvez commencer à gérer n'importe quelle instance cloud ou logiciel réseau sans modifier l'image de base.

D. Pourquoi avons-nous besoin de Kubernetes ?

Les véritables applications de production s'étendent sur plusieurs conteneurs, des conteneurs qui doivent être déployés sur différents hôtes serveur. La sécurité des conteneurs comporte plusieurs couches et peut être complexe. C'est précisément à ce niveau qu'intervient la technologie Kubernetes. Kubernetes nous offre les outils d'orchestration et de gestion requis pour déployer des conteneurs, à grande échelle, pour ces charges de travail. Les fonctionnalités d'orchestration de Kubernetes nous permettent de créer des services applicatifs sur plusieurs conteneurs, de planifier l'exécution de ces conteneurs dans un cluster, de les mettre à l'échelle et de gérer leur intégrité au fil du temps. Avec Kubernetes, nous pouvons prendre des mesures concrètes pour améliorer la sécurité informatique.

Kubernetes doit pouvoir s'intégrer aux services de mise en réseau, de stockage, de sécurité, de télémétrie, entre autres, pour fournir une infrastructure de conteneurs complète. Une application de conteneurs Linux rudimentaire les traite comme des machines virtuelles rapides et efficaces. Mais une fois que nous évoluons vers un environnement de production et multiplions le nombre d'applications, surgit le besoin de plusieurs conteneurs colocalisés qui fonctionnent ensemble pour fournir des services individuels. Le nombre de conteneurs dans notre environnement augmente alors significativement et l'accumulation de ces conteneurs complexifie notre infrastructure.

Kubernetes vous permet de résoudre de nombreux problèmes courants liés à la prolifération des conteneurs en les triant au sein d'un « pod ». Ces pods ajoutent une couche d'abstraction aux groupes de conteneurs, ce qui nous aide à planifier les charges de travail et à

MISE EN ŒUVRE

fournir les services nécessaires (réseau, stockage, etc.) a ces conteneurs. D'autres composants de Kubernetes nous aident à équilibrer la charge sur ces pods et à nous assurer que nous disposons de suffisamment de conteneurs pour exécuter nos charges de travail.

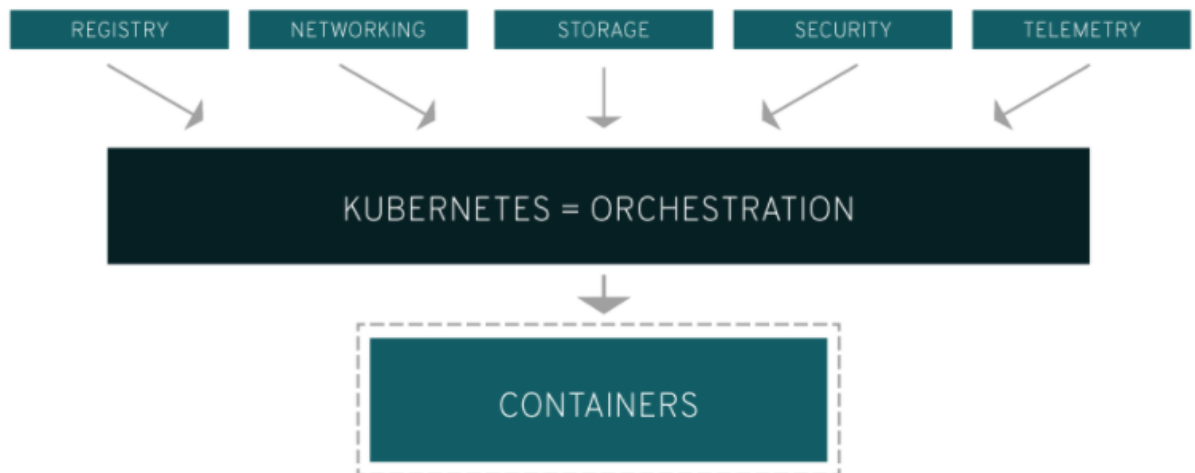


Figure 5-4: Kubernetes

Que pouvons-nous faire avec Kubernetes ?

Le principal avantage de la technologie Kubernetes est qu'elle offre une plateforme dédiée à la planification et à l'exécution de conteneurs sur des clusters de machines physiques ou virtuelles.

Autrement dit, elle nous aide à mettre en œuvre et à utiliser une infrastructure de conteneurs dans nos environnements de production. Kubernetes vise avant tout l'automatisation des tâches opérationnelles, c'est pourquoi cette technologie nous permet d'effectuer un grand nombre de tâches que d'autres plateformes d'applications et systèmes de gestion nous permettent déjà de faire, mais pour nos conteneurs.

Avec Kubernetes, nous pouvons :

- Orchestrer des conteneurs sur plusieurs hôtes ;
- Optimiser l'utilisation de notre matériel afin de maximiser les ressources requises pour l'exécution de nos applications d'entreprise ;

- Contrôler et automatiser les déploiements et mises à jour d'applications ;
- Monter et ajouter des systèmes de stockage pour exécuter des applications avec état;
- Mettre à l'échelle des applications conteneurisées et leurs ressources à la volée;
- Gérer des services de façon déclarative et garantir ainsi que les applications déployées s'exécutent toujours de la manière dont nous les avons déployées ;
- Vérifier l'intégrité de nos applications et les réparer automatiquement grâce au placement, au démarrage, à la réplication et à la mise à l'échelle automatiques.

Terminologies Kubernetes

A l'instar de toute technologie, Kubernetes dispose d'une terminologie propre. Voyons les termes les plus courants.

Maître : serveur qui contrôle les nœuds Kubernetes, sur lequel toutes les tâches sont assignées.

Nœuds : machines qui exécutent les tâches qui leur sont assignées. Ces nœuds sont contrôlés par le serveur maître Kubernetes.

Pod : groupe d'un ou de plusieurs conteneurs déployés sur un seul nœud. Tous les conteneurs d'un pod partagent une même adresse IP, un même IPC, un même nom d'hôte et d'autres ressources. Les pods séparent le réseau et le stockage du conteneur sous-jacent. Ainsi, nous pouvons déplacer nos conteneurs au sein du cluster très simplement.

Contrôleur de réplication : composant qui vérifie le nombre de copies identiques d'un pod qui doivent s'exécuter quelque part dans le cluster.

Service : élément qui dissocie les définitions de tâche des pods. Les proxys de service de Kubernetes transfèrent automatiquement les requêtes de service vers le pod pertinent, même si celui-ci a été précédemment déplacé ou remplacé.

Kubelet : service exécute sur des nœuds qui lit les manifestes du conteneur pour s'assurer que les conteneurs définis ont démarré et fonctionnent.

Kubectl : outil de configuration en ligne de commande de Kubernetes.

Aperçu du rôle de Kubernetes dans notre infrastructure

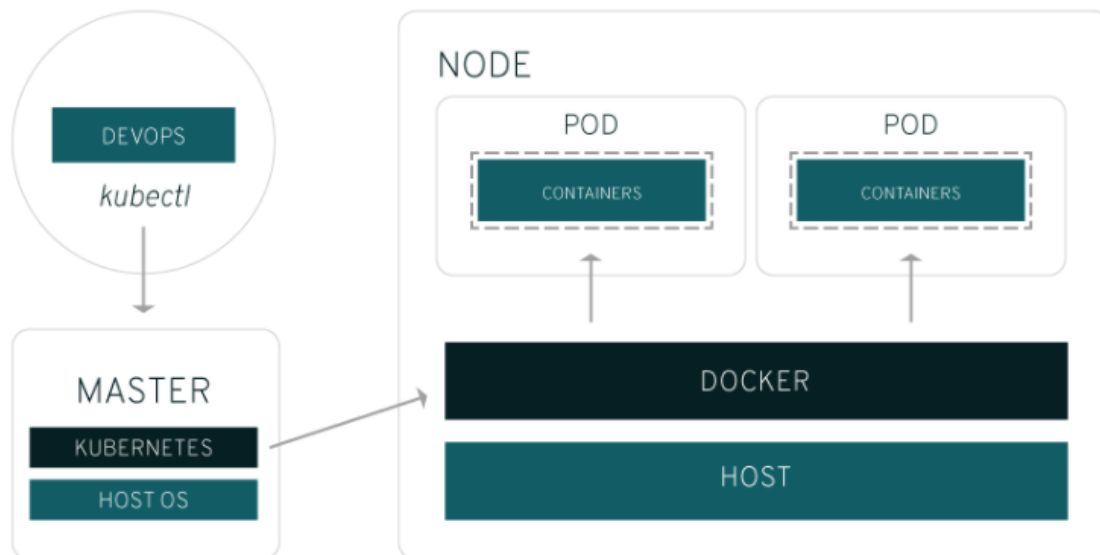


Figure 5-5: Aperçu du rôle de Kubernetes dans notre infrastructure

Kubernetes s'exécute au-dessus d'un système d'exploitation et interagit avec les pods de conteneurs qui s'exécutent sur les nœuds. Le serveur maître Kubernetes reçoit les commandes de la part d'un administrateur (ou d'une équipe DevOps) et relaie ces instructions aux nœuds qui lui sont subordonnés. Ce système de transfert fonctionne avec une multitude de services et choisit automatiquement le nœud le plus adapté pour chaque tâche. Il alloue ensuite les ressources aux pods désignés dans ce nœud pour qu'ils exécutent la tâche requise.

Aussi, du point de vue de l'infrastructure, la gestion des conteneurs reste très similaire à celle que nous connaissons. Nous contrôlons simplement les conteneurs à un niveau plus élevé et plus efficacement, sans avoir à gérer chaque conteneur ou nœud de manière individuelle. Certaines tâches manuelles nous incombent encore, mais il s'agit surtout d'assigner un serveur maître Kubernetes, de définir des nœuds et des pods.

La technologie Docker continue de jouer son rôle. Lorsque Kubernetes planifie un pod dans un nœud, le kubelet de ce nœud ordonne à Docker de lancer les conteneurs spécifiques. Le kubelet collecte ensuite en continu le statut de ces conteneurs via Docker et rassemble ces

informations sur le serveur maître. Docker transfère les conteneurs dans le nœud et démarre/arrête ces conteneurs, comme d'habitude. La différence est l'origine des ordres : ils proviennent d'un système automatisé et non plus d'un administrateur qui assigne manuellement des tâches à tous les nœuds pour chaque conteneur.

E. Pourquoi répéter le déploiement sur Openshift ?

Red Hat OpenShift est une plateforme Kubernetes de pointe pour les entreprises qui offre une expérience digne du cloud, partout où elle est déployée. Que ce soit dans le cloud, sur site ou en périphérie, Red Hat OpenShift vous donne la possibilité de choisir où créer, déployer et exécuter vos applications grâce à une expérience cohérente. Cette plateforme assure l'exploitation automatisée de toute la pile et le provisionnement en libre-service, ce qui permet aux équipes de collaborer pour concrétiser les projets de développement plus efficacement.

C'est notamment le produit utilisé dans l'entreprise où j'ai passé ce période d'étude et de recherche. De plus le déploiement automatique sur cette plateforme n'a jamais été fait ou du moins ne l'est pas à notre connaissance.

OpenShift fournit une plateforme complète pour les applications traditionnelles et cloud-native, ce qui permet de les exécuter partout. Basée sur Red Hat Enterprise Linux et compatible avec Red Hat Ansible Automation Platform, elle permet de mettre en œuvre l'automatisation à l'intérieur et à l'extérieur des clusters Kubernetes.

5.3. Activités sécurité à forte valeur ajoutée intégrables dans un pipeline devops

Le but de cette partie n'est pas d'être exhaustif, mais d'évoquer des pistes qui permettent d'améliorer significativement le niveau de sécurité sans y consacrer un temps ou un budget trop conséquent. Les recommandations s'appliquent que l'on produise du code pour le Web ou des

clients lourds comme les applications mobiles. Mais l'OWASP ayant dans son nom et ses racines un lien fort avec le monde du Web, nous aurons tendance à mettre plus en avant ce domaine.

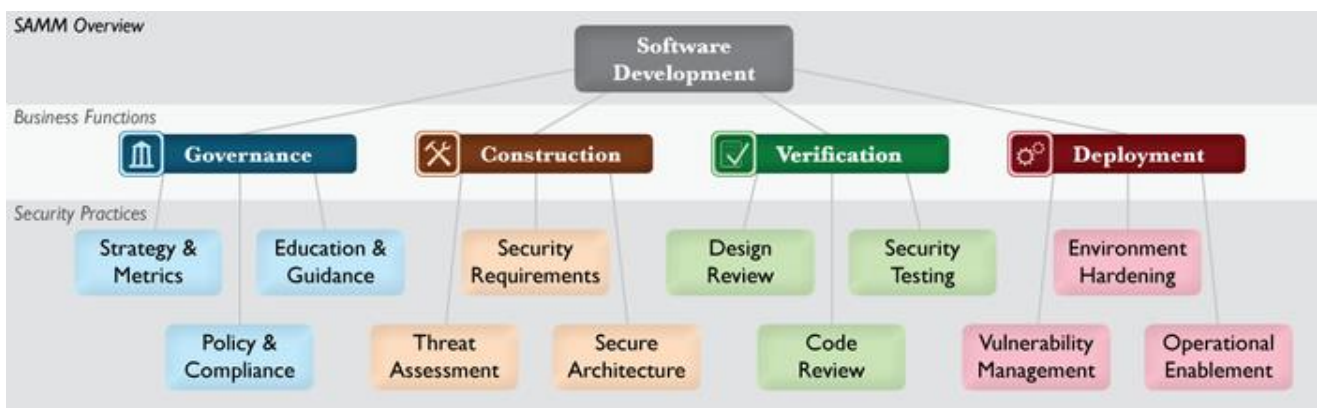


Figure 5-6: Les pratiques de sécurité recommandées par l'OWASP dans son modèle de maturité

A. Tests sécurité : tests d'intrusion automatisés

Procéder à des tests d'intrusion est la pratique la plus répandue et une section y est dédiée dans chaque numéro de ce magazine. Lorsque qu'il s'agit d'une application web ou mobile s'appuyant sur des services web, des outils automatisés existent et sont souvent utilisés par les professionnels pour dégrossir le travail. Certains de ces outils sont gratuits et accessibles à un adolescent sans grande connaissance du domaine.

Il est alors envisageable de les inclure à moindre coût dans l'environnement d'intégration continue. Cela permet d'avoir une première couverture contre les attaques classiques du Web, ce qui devrait décourager un adolescent et l'inciter à changer de cible. Les sociétés de test d'intrusion mandatées auront plus de temps à consacrer à des scénarios élaborés. Et surtout cela réduit la probabilité de se faire pirater à cause d'une faute élémentaire qui pourrait être catastrophique pour la réputation de l'entreprise.

Deux outils vont être présentés, les critères de sélection étant par ordre d'importance :

- la facilité d'intégration dans un environnement de déploiement continu ;
- le fait que les failles identifiées soient les plus répandues ;
- la facilité d'interprétation des résultats pour un développeur qui n'est pas expert en sécurité ;
- la gratuité.

Ils sont certainement déjà connus des lecteurs assidus ou considères élémentaires ou pas assez efficaces par les professionnels du domaine.

OWASP ZAP

Le projet ZAP d'OWASP est un outil graphique intègre de tests d'intrusion pour trouver des vulnérabilités dans les applications. Comme explicitement mentionné sur le site web, il a été conçu pour pouvoir être utilisé par des développeurs et testeurs fonctionnels, mais aussi par des personnes plus expérimentées. Il permet ainsi de faire des investigations manuellement.

L'utilisation la plus élémentaire est simplement de fournir une URL et de cliquer sur Attaquer. Puis d'attendre que le scanner teste une série de règles après avoir construit la cartographie des URLs du site. Il est particulièrement efficace sur la détection de failles XSS, en étant capable de tenir compte du comportement dynamique d'une page exécutant de nombreuses requêtes Ajax.

C'est un parfait candidat à l'intégration dans un pipeline DevOps puisqu'il dispose d'APIs permettant de piloter un scan. C'est ce qui est réalisé par le plugin ZA Proxy pour Jenkins. On peut entre autres définir le type de scanner (XSS, CSRF, SQLi, etc.), les domaines sur lesquels rester (pour éviter de suivre des liens externes) et surtout exporter le rapport sous format XML. Mais aussi lui préciser ce qui a déjà été analysé au préalable.

SQLMAP

Sqlmap est un outil avancé en ligne de commandes permettant de détecter et exploiter des failles de type injection SQL. Il est notamment capable de détecter le type de base de données et d'en exporter l'intégralité de son contenu si elle est mal configurée et qu'il y a une requête exploitable. A ce sujet, il gère totalement les six différentes techniques d'injection connues à ce jour. Il est fréquemment utilisé par des groupes de pirates, l'option de passer par le réseau Tor étant attrayante pour les usages frauduleux.

L'utilisation la plus élémentaire consiste à lui fournir une URL d'un site web contenant un paramètre et il testera alors si ce dernier est injectable. Les résultats sont affichés directement dans la console en ligne de commandes.

Sqlmap est un complément très utile à ZAP, ce dernier étant moins efficace sur la détection des injections SQL et générant un nombre conséquent de faux positifs pour cette catégorie. Comme sqlmap gère en entrée une liste d'URL (e.g. sitemap.xml), on peut lui fournir la cartographie qui a été découverte par ZAP. Cela peut facilement se configurer dans une tâche Jenkins par exemple.

Il faut par contre parser les résultats qui ne sont pas structurés.

Défis dans la gestion de ces outils

Armés de ces deux outils automatisés, nous possédons une couverture raisonnable pour une application web contre les attaques SQLi et XSS, respectivement première et troisième du Top 10 OWASP.

Le premier défi est celui des temps de scan, Sqlmap et ZAP pouvant prendre de quelques minutes à plusieurs heures selon la complexité de l'application. Il est donc difficilement envisageable de faire une vérification à chaque fois que du code est envoyé dans le SCM. Même si cela serait l'idéal en notifiant en quasi temps réel au développeur la vulnérabilité insérée et en pouvant la corriger de suite à moindres frais. Si l'on ne produit pas plusieurs releases par

jour, un scan la nuit sur la branche de développement (au sens Git workflow) est un bon compromis et permet de ne pas écrouler l'infrastructure la journée quand les équipes travaillent. Et au minimum il faut que chaque version promue soit vérifiée.

Le second défi est certainement le plus important : il est question d'avoir la capacité de traiter toutes les informations remontées par les outils. Il ne s'agit plus de gérer quelques remarques particulières évoquées dans un seul audit concis accompagné de ses recommandations concrètes. Il va éventuellement falloir analyser des centaines de points tous les jours, dont une part significative est constituée de faux positifs. Il est alors essentiel :

- D'automatiser le traitement des résultats ;
- D'être capable de filtrer les points déjà traités et les faux positifs ;
- D'identifier uniquement une vulnérabilité pour la suivre dans le temps ;
- D'avoir un canal de correction efficace avec le développeur : la stratégie « une vulnérabilité = une entrée dans la solution de bugtracking » va rapidement mener à des coûts de gestion considérables.

Aujourd'hui cela nécessite la mise en place d'outils ad hoc pour remplir ces critères. Mais si l'on parle d'une application web implémentée correctement vis-à-vis de l'OWASP Top 10 ou qui a subi un audit externe, une approche moins industrielle est encore viable puisqu'il ne devrait pas y avoir trop de points identifiés (hors faux positifs). Si ce n'est pas le cas, il est certainement plus judicieux de commencer par corriger la sécurité de l'application ! Par exemple lors d'un sprint de consolidation technique si l'on suit une méthodologie agile.

Enfin, le dernier défi est de ne pas basculer dans un mode de correction suivant la loi du moindre effort. En effet, beaucoup de vulnérabilités ne sont plus exploitables en adaptant la configuration ou avec une rustine minimaliste. Mais il faut bien garder à l'esprit que ces

vulnérabilités sont des erreurs d'implémentation comme tout bug, et qu'elles doivent donc être traitées avec la même cohérence. C'est-à-dire au minimum par l'écriture d'un test unitaire démontrant l'efficacité de la modification. Il est aussi recommandé de faire tourner ces outils dans un environnement volontairement laxiste, afin de rendre l'exploitation de ces erreurs-là plus facile possible et ainsi d'améliorer le taux de détection.

B. Tests sécurité : tests unitaires de scénarios d'abus

Après avoir testé la sécurité de l'application depuis l'extérieur, processus assez fréquent, il est très important de la valider aussi depuis ses entrailles. C'est ce qui permet de s'assurer de sa robustesse sur le long terme, les vulnérabilités finissant par être identifiées tôt ou tard. A ce titre, toute section sensible du code doit faire l'objet de tests unitaires exhaustifs. Cela inclut notamment :

- la logique d'authentification ;
- les mécanismes de contrôle d'accès ;
- les fonctionnalités de paiement ;
- les opérations cryptographiques.

Il ne suffit pas de tester que le cas nominal, mais aussi tous les cas d'erreur. Et de vérifier que les logs applicatifs ramènent le niveau de détail suffisant pour comprendre qu'elle est précisément la raison de l'échec, sans pour autant noyer le message utile dans des centaines de lignes très techniques (ce qui est souvent le problème des exceptions en Java).

C. En quoi consiste la sécurité des conteneurs ?

La sécurité des conteneurs correspond à la protection de l'intégrité des conteneurs, des applications qu'ils renferment à l'infrastructure sur laquelle ils s'appuient. Elle doit être intégrée et continue. Pour assurer la sécurité continue des conteneurs, il faut généralement :

- Sécuriser le pipeline de conteneurs et l'application ;
- Sécuriser l'environnement de déploiement des conteneurs et l'infrastructure ;
- Exploiter les outils de sécurité de l'entreprise tout en respectant ou en améliorant les politiques de sécurité existantes.

Les conteneurs sont largement utilisés, car ils facilitent la création, la mise en paquets et la promotion d'une application ou d'un service et de toutes leurs dépendances, tout au long du cycle de vie et dans différents environnements et cibles de déploiement. Néanmoins, la sécurité des conteneurs peut être difficile à mettre en œuvre. En effet, les politiques de sécurité et les listes de contrôle statiques ne s'adaptent pas aux conteneurs de l'entreprise. La chaîne d'approvisionnement nécessite davantage de services en matière de politiques de sécurité. Les équipes doivent trouver un équilibre entre les besoins de mise en réseau et les impératifs de gouvernance liés aux conteneurs.

Enfin, les outils de création et d'exécution doivent être dissociés des services.

Sécurisation du pipeline de conteneurs

Collecte d'images

Les conteneurs sont constitués de couches de fichiers, que la communauté d'utilisateurs appelle des « images de conteneurs ». L'image de base est la plus importante en matière de sécurité, car elle sert de point de départ pour la création d'images dérivées. La sécurité des conteneurs commence donc par l'identification de sources fiables pour les images de base. Toutefois, même avec des images fiables, l'ajout d'applications et les changements de configuration introduisent de nouvelles variables. Lorsque nous ajoutons du contenu externe pour créer nos applications, nous devons garder à l'esprit les principes de gestion proactive de contenu.

Avant de collecter des images de conteneurs, les questions suivantes doivent être posées :

- Les images de conteneurs sont-elles signées et issues de sources fiables ?
- Les couches d'exécution et du système d'exploitation sont-elles à jour ?
- A quelle fréquence les conteneurs seront-ils mis à jour et combien de temps dureront ces opérations ?
- Des problèmes connus ont-ils été identifiés et comment seront-ils suivis ?

Gestion des accès

Une fois la collecte terminée, l'étape suivante consiste à gérer l'accès à ces images et leur partage avec l'équipe qui les utilise. Cela implique de protéger les images que nous téléchargeons et celles que nous créons. L'utilisation d'un registre privé nous permet de contrôler l'accès en fonction des rôles et facilite la gestion du contenu grâce à l'affectation de métadonnées au conteneur. Les métadonnées fournissent des informations qui permettent d'identifier et de suivre des vulnérabilités connues. Avec un registre privé, nous pouvons également automatiser et affecter des politiques aux images de conteneurs stockées, tout en limitant les risques d'erreurs humaines qui peuvent être à l'origine de vulnérabilités dans nos conteneurs.

Pour définir la méthode de gestion des accès, les questions suivantes doivent être posées :

- Quels contrôles d'accès basés sur les rôles faut-il mettre en place pour gérer les images de conteneurs ?
- Des fonctionnalités de balisage sont-elles prévues pour faciliter le tri des images ? Les images peuvent-elles être balisées pour indiquer qu'elles ont été approuvées uniquement pour le développement, puis pour les tests et enfin pour la production ?
- Le registre fournit-il des métadonnées visibles qui permettent de suivre des vulnérabilités connues ?

- Le registre peut-il être utilisé pour affecter et automatiser une politique (par exemple, la vérification de signatures, l'analyse de code, etc.) ?

Intégration de tests de sécurité et automatisation du déploiement

La dernière étape du pipeline est le déploiement. Une fois les versions créées, nous devons les gérer conformément aux normes du secteur. Pour cela, nous devons comprendre comment automatiser des politiques qui permettent de signaler les versions présentant des problèmes de sécurité, notamment en cas de détection de nouvelles vulnérabilités. L'application de correctifs n'étant pas aussi efficace qu'une reconstruction des conteneurs, l'intégration de tests de sécurité doit inclure la définition de politiques qui déclenchent des reconstructions automatiques. La première étape de cette démarche consiste à exécuter des outils d'analyse des composants pour suivre et signaler les problèmes. La seconde étape vise à mettre en place des outils qui permettent des déploiements automatisés et basés sur des politiques.

Au moment de l'intégration des tests de sécurité et de l'automatisation du déploiement, la question suivante doit être posée :

- Comment éviter l'application de correctifs aux conteneurs en cours d'exécution et prévoir à la place d'utiliser des déclencheurs pour reconstruire et remplacer les conteneurs au moyen de mises à jour automatiques ?

Protection de l'infrastructure

Le système d'exploitation hôte fournit une autre couche de sécurité aux conteneurs en les isolant, c'est pourquoi nous avons besoin d'un système d'exploitation hôte qui puisse leur offrir une isolation maximale. Cette isolation joue un rôle primordial dans la protection de l'environnement de nos déploiements de conteneurs. Le système d'exploitation hôte est activé au moyen d'un environnement d'exécution de conteneur, qui est idéalement géré par un système d'orchestration. Pour rendre notre plateforme de conteneurs résiliente, nous allons utiliser des

MISE EN ŒUVRE

espaces de noms réseau qui isolent les applications et les environnements, et associé le système de stockage à l'aide de mécanismes de montage sécurisés. Une solution de gestion des API doit comprendre l'authentification et l'autorisation, l'intégration LDAP, des contrôles d'accès des points de terminaison et la limitation du débit.

Pour mettre en œuvre une protection efficace de l'infrastructure de conteneurs, les questions suivantes doivent être posées :

- Quels conteneurs doivent pouvoir accéder à d'autres conteneurs ? Comment s'effectue leur détection ?
- Comment contrôler l'accès aux ressources partagées et leur gestion (par exemple, le réseau et le stockage) ?
- Comment gérer les mises à jour de l'hôte ? Tous les conteneurs devront-ils être mis à jour simultanément ?
- Comment surveiller l'intégrité des conteneurs ?
- Comment faire évoluer automatiquement la capacité des applications pour répondre à la demande ?

5.4. Installation et configuration des solutions de l'architecture

A. Installation Jenkins

```
[jenkins@localhost ~]$ vim /etc/yum.repos.d/jenkins.repo
[jenkins@localhost ~]$ cat /etc/yum.repos.d/jenkins.repo
name=Jenkins
baseurl=http://pkg.jenkins.io/redhat
gpgcheck=1

[jenkins@localhost ~]$ sudo yum install jenkins
[sudo] password for jenkins:
Updating Subscription Management repositories.
Last metadata expiration check: 3:25:56 ago on Wed 21 Dec 2022 06:21:12 AM GMT.
Package jenkins-2.380-1.1.noarch is already installed.
Dependencies resolved.
=====
Package                Architecture      Version           Repository        Size
=====
Upgrading:
jenkins                noarch            2.383-1.1        jenkins           94 M
=====
Transaction Summary
=====
Upgrade 1 Package

Total download size: 94 M
Is this ok [y/N]:
```

Figure 5-7: Ajout du répertoire et installation de Jenkins

```
[jenkins@localhost ~]$ chkconfig jenkins on
Note: Forwarding request to 'systemctl enable jenkins.service'.
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
You do not have enough privileges to perform this operation.
[jenkins@localhost ~]$
```

Figure 5-9: Activation du service de Jenkins

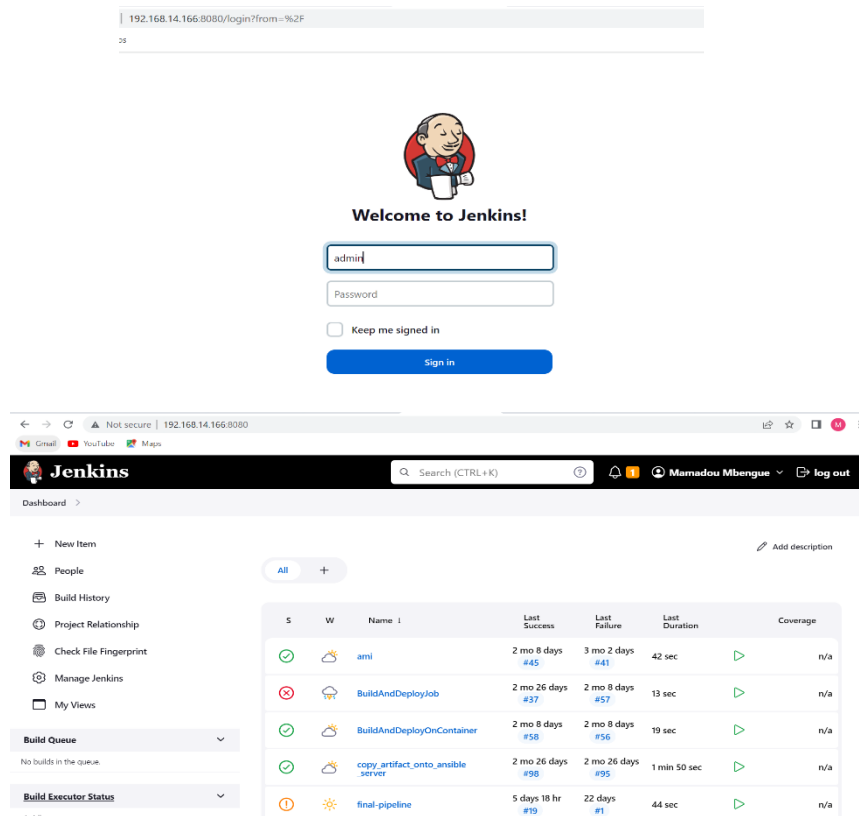


Figure 5-8: Interface Graphique de Jenkins

MISE EN ŒUVRE

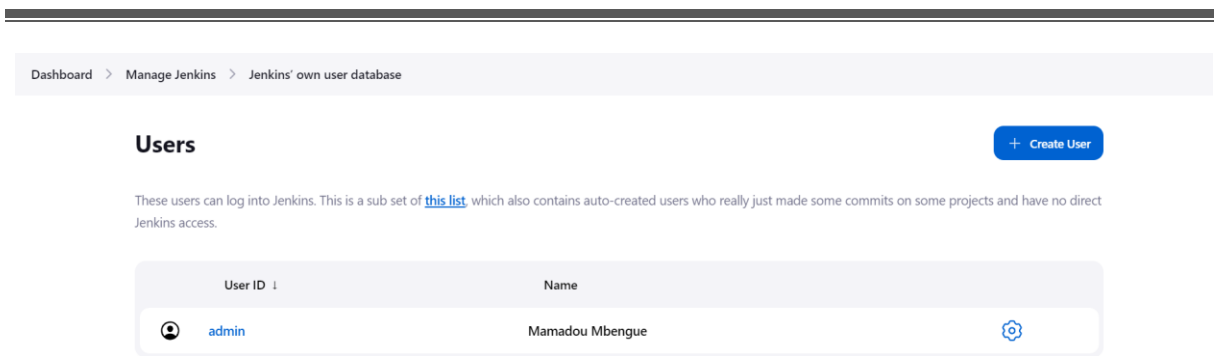


Figure 5-11: Création de User

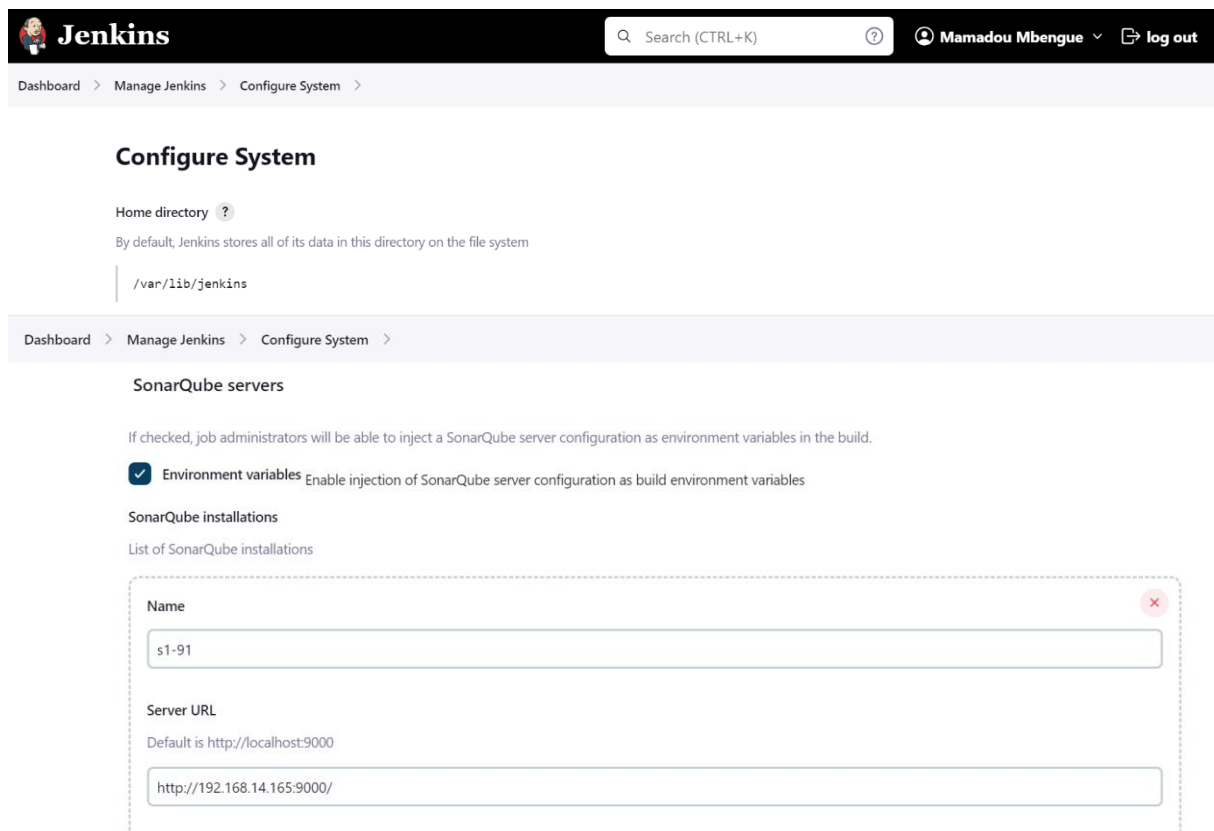


Figure 5-10: Intégration avec Sonarqube

MISE EN ŒUVRE

SSH Server

Name ?

ansible

Hostname ?

192.168.14.167

Username ?

ansible

SSH Server

Name ?

master_node

Hostname ?

192.168.14.152

Username ?

sonar

SSH Server

Name ?

dockerhost

Hostname ?

192.168.14.144

Username ?

dockeradmin

Remote Directory ?

Figure 5-12: Intégration des serveur (ansible, dockerhost, et du master Node)

MISE EN ŒUVRE

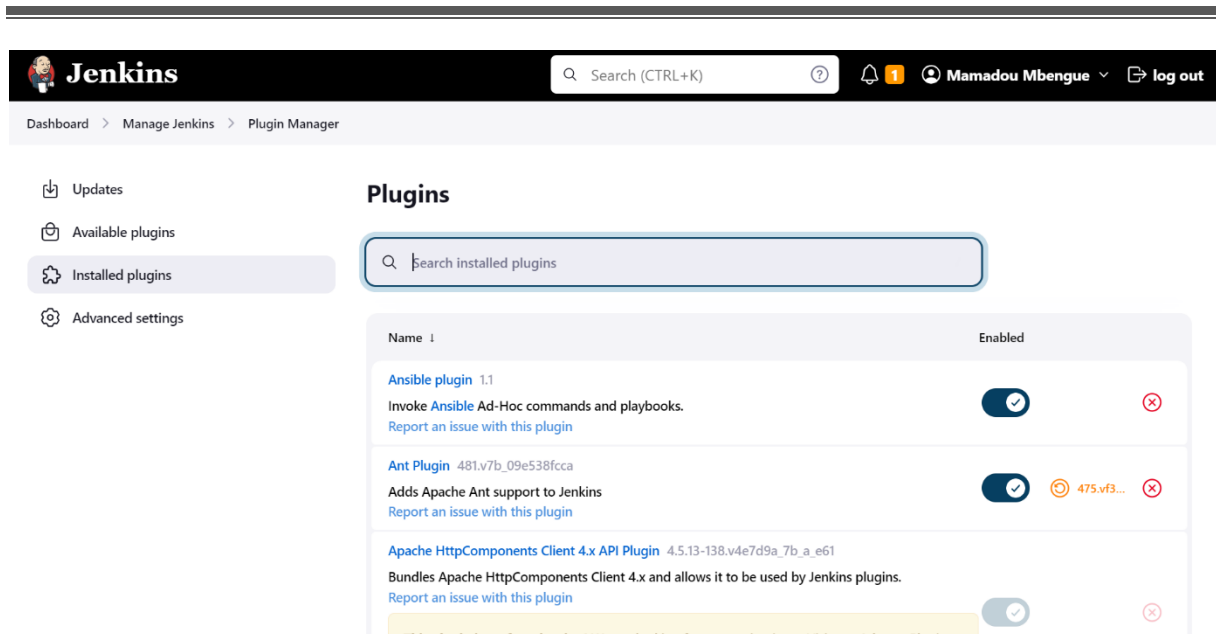


Figure 5-13: Ajouter des plugins d'intégration

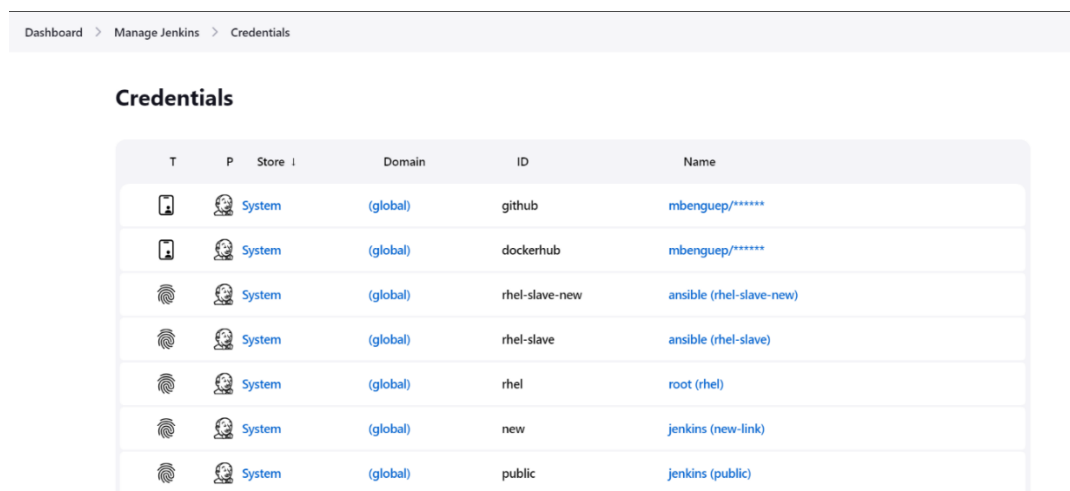


Figure 5-14: Intégration des comptes à Utiliser

B. Installation Kubernetes

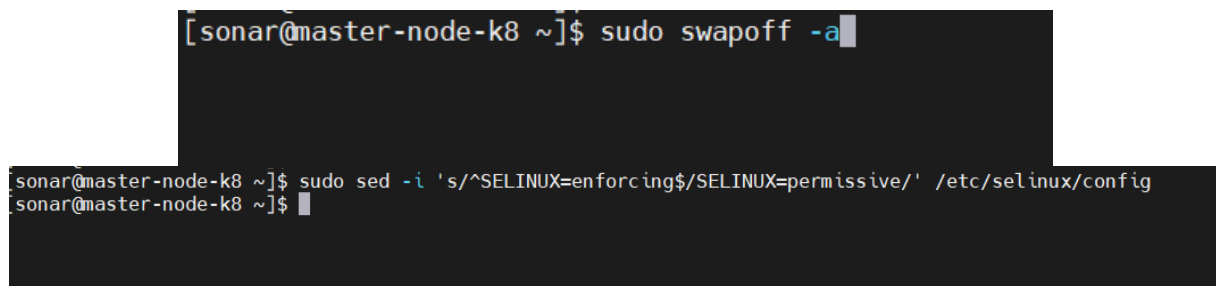


Figure 5-15: Préparation de l'environnement

```
[sonar@master-node-k8 ~]$ sudo firewall-cmd --permanent --add-port=6443/tcp & \
> sudo firewall-cmd --permanent --add-port=2379-2380/tcp & \
> sudo firewall-cmd --permanent --add-port=10250/tcp & \
> sudo firewall-cmd --permanent --add-port=10251/tcp & \
> sudo firewall-cmd --permanent --add-port=10252/tcp & \
> sudo firewall-cmd --reload
[sonar@master-node-k8 ~]$ sudo vim /etc/hosts
#127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
#::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.14.163 worker-node-2-k8
192.168.14.164 worker-node-1-k8
192.168.14.152 master-node-k8
```

Figure 5-16: Ajout des ports et des noeuds

```
[sonar@master-node-k8 ~]$ sudo vi /etc/modules-load.d/k8s.conf
overlay
br_netfilter

[sonar@master-node-k8 ~]$ sudo vi /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
```

Figure 5-17: Configuration du Sysctl

```
[sonar@master-node-k8 ~]$ sudo sysctl --system
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
kernel.yama.ptrace_scope = 0
* Applying /usr/lib/sysctl.d/50-coredump.conf ...
kernel.core_pattern = |/usr/lib/systemd/systemd-coredump %P %u %g %s %t %c %h %e
kernel.core_pipe_limit = 16
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1
kernel.kptr_restrict = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.promote_secondaries = 1
net.core.default_qdisc = fq_codel
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
* Applying /usr/lib/sysctl.d/50-libkcap-optmem-max.conf ...
net.core.optmem_max = 81920
* Applying /usr/lib/sysctl.d/50-pid-max.conf ...
kernel.pid_max = 4194304
* Applying /usr/lib/sysctl.d/60-libvirtd.conf ...
fs.aio-max-nr = 1048576
* Applying /usr/lib/sysctl.d/60-qemu-postcopy-migration.conf ...
* Applying /etc/sysctl.d/99-sonarqube.conf ...
vm.max_map_count = 262144
fs.file-max = 65536
* Applying /etc/sysctl.d/99-sysctl.conf ...
vm.max_map_count = 262144
fs.file-max = 65536
* Applying /etc/sysctl.d/k8s.conf ...
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
net.bridge.bridge-nf-call-ip6tables = 1
* Applying /etc/sysctl.conf ...
vm.max_map_count = 262144
fs.file-max = 65536
[sonar@master-node-k8 ~]$ export VERSION=1.21
[sonar@master-node-k8 ~]$
```

Figure 5-18: Activation des config

MISE EN ŒUVRE

```
[sonar@master-node-k8 ~]$ sudo curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo https://download.opensuse.org/repositories/dev
el:kubic:libcontainers:stable:cri-o:$VERSION/CentOS_8/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
100 425 100 425 0 0 1190 0 --:--:-- --:--:-- --:--:-- 1190
100 426 100 426 0 0 961 0 --:--:-- --:--:-- --:--:-- 961
100 427 100 427 0 0 884 0 --:--:-- --:--:-- --:--:-- 884
100 428 100 428 0 0 693 0 --:--:-- --:--:-- --:--:-- 693
100 429 100 429 0 0 611 0 --:--:-- --:--:-- --:--:-- 611
100 381 100 381 0 0 478 0 --:--:-- --:--:-- --:--:-- 478
[sonar@master-node-k8 ~]$
[sonar@master-node-k8 ~]$ sudo dnf install cri-o
Updating Subscription Management repositories.
Last metadata expiration check: 0:00:45 ago on Fri 16 Dec 2022 06:29:10 AM EST.
Dependencies resolved.
=====
Package                Architecture      Version           Repository          Size
=====
Installing:
cri-o                  x86_64            1.21.7-6.1.el8    devel_kubic_libcontainers_stable_cri-o_1.21 23 M
=====
Transaction Summary
=====
Install 1 Package

Total download size: 23 M
Installed size: 117 M
Is this ok [y/N]: █

[sonar@master-node-k8 ~]$ sudo systemctl enable --now cri-o
Created symlink /etc/systemd/system/multi-user.target.wants/crio.service → /usr/lib/systemd/system/crio.service.
[sonar@master-node-k8 ~]$ █

[sonar@master-node-k8 ~]$ sudo vi /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
█

[sonar@master-node-k8 ~]$ sudo dnf install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Updating Subscription Management repositories.
Last metadata expiration check: 0:30:38 ago on Fri 16 Dec 2022 06:29:10 AM EST.
Package kubelet-1.25.2-0.x86_64 is already installed.
Package kubeadm-1.25.2-0.x86_64 is already installed.
Package kubectl-1.25.2-0.x86_64 is already installed.
Dependencies resolved.
=====
Package                Architecture      Version           Repository          Size
=====
Upgrading:
kubeadm                x86_64            1.26.0-0          kubernetes          10 M
kubectl                x86_64            1.26.0-0          kubernetes          11 M
kubelet                x86_64            1.26.0-0          kubernetes          22 M
=====
Transaction Summary
=====
Upgrade 3 Packages

Total download size: 43 M
Download Packages:
(1/3): da58cbf31a0337a968e5a06cfc00e420cc2df8930ea817cd2a4227bd81d49-kubeadm-1.26.0-0.x86_64.rpm 2.7 MB/s | 10 MB 00:03
(2/3): 23e112935127da08ffdc32c392cbf62346305ee97ba6c5d070cda422945e4ff-kubectl-1.26.0-0.x86_64.rpm 2.6 MB/s | 11 MB 00:04
(3/3): 9be8590c2de60e249f40726e979a3a7a046320079bc41d330834de74f5399383-kubelet-1.26.0-0.x86_64.rpm 4.2 MB/s | 22 MB 00:05
-----
Total 8.2 MB/s | 43 MB 00:05
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing                : 1/1
  Running scriptlet: kubelet-1.26.0-0.x86_64 1/1
  Upgrading           : kubelet-1.26.0-0.x86_64 1/6
  Upgrading           : kubectl-1.26.0-0.x86_64 2/6

[sonar@master-node-k8 ~]$ sudo systemctl enable --now kubelet
[sonar@master-node-k8 ~]$ █
```

Figure 5-19: Ajout du répertoire et installation des paquets

MISE EN ŒUVRE

```
[sonar@master-node-k8 ~]$ sudo kubeadm init --pod-network-cidr=192.168.10.0/16
[init] Using Kubernetes version: v1.26.0
[preflight] Running pre-flight checks
[WARNING] FirewallD: firewallD is active, please ensure ports [6443 10250] are open or your cluster may not function correctly
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[certs] Using certificateDir folder '/etc/kubernetes/pki'
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes.kubernetes.default.kubernetes.default.svc.kubernetes.default.svc.cluster.local master-node-k8] and IPs [10.96.0.1 192.168.14.152]
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [localhost master-node-k8] and IPs [192.168.14.152 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost master-node-k8] and IPs [192.168.14.152 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 4.501618 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
```

Figure 5-21: Configuration du réseau

```
[sonar@master-node-k8 ~]$ kubectl get nodes -o wide
NAME                STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION        CONTAINER-RUNTIME
master-node-k8      Ready    control-plane   28m   v1.26.0   192.168.14.152   <none>         Red Hat Enterprise Linux 8.6 (Ootpa)   4.18.0-372.26.1.el8_6.x86_64   cri-o://1.21.7

[sonar@master-node-k8 ~]$ kubectl get nodes -o wide
NAME                STATUS    ROLES    AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION        CONTAINER-RUNTIME
master-node-k8      Ready    control-plane   4h47m   v1.26.0   192.168.14.152   <none>         Red Hat Enterprise Linux 8.7 (Ootpa)   4.18.0-372.26.1.el8_6.x86_64   cri-o://1.21.7
worker-node-1-k8    Ready    <none>      19m   v1.26.0   192.168.14.163   <none>         Red Hat Enterprise Linux 8.7 (Ootpa)   4.18.0-425.3.1.el8_8.x86_64   cri-o://1.21.7
worker-node-2-k8    Ready    <none>      19m   v1.26.0   192.168.14.164   <none>         Red Hat Enterprise Linux 8.7 (Ootpa)   4.18.0-425.3.1.el8_8.x86_64   cri-o://1.21.7
```

Figure 5-23: Liste des nœuds

```
[sonar@worker-node-2-k8 ~]$ sudo kubeadm join 192.168.14.152:6443 --token a7yfir.464o7ikhk20rfdbn \
--discovery-token-ca-cert-hash sha256:a32b4be692150326c07a15ff118bbb5589b7f129f774521360176333beba2f8d
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
etcdctl snapshot save /etc/kubernetes/backup/etcd-backup-2024-08-08-14:15:00 --operator-username=operator --operator-password=operator
```

Figure 5-22: Intégration des nœuds worker

```
installation.operator.tigera.io/default created
apiserver.operator.tigera.io/default created
[sonar@master-node-k8 ~]$ kubectl get pods -n calico-system
NAME                                READY    STATUS    RESTARTS   AGE
calico-kube-controllers-67df98bdc8-4g5jj  1/1      Running    0           82s
calico-node-wqwtl                      1/1      Running    0           82s
calico-typha-7c7d449594-4k42p          1/1      Running    0           82s
[sonar@master-node-k8 ~]$
```

Figure 5-20: Configuration du système de Calico

C. Installation Ansible

```
[ansible@ansible-server ~]$ sudo yum install python3-pip
Updating Subscription Management repositories.
Last metadata expiration check: 2:28:28 ago on Wed 21 Dec 2022 01:12:40 PM GMT.
Package python3-pip-9.0.3-22.el8.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ansible@ansible-server ~]$ █
[ansible@ansible-server ~]$ useradd ansible
```

Figure 5-24: Installation de Python et Création de l'utilisateur

```
[ansible@ansible-server ~]$ echo "ansible ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
[ansible@ansible-server ~]$ █
[ansible@ansible-server ~]$ █
[ansible@ansible-server ~]$ sudo sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
[ansible@ansible-server ~]$ █
```

Figure 5-25: Configuration des Privilèges pour l'utilisateur

```
[ansible@ansible-server ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansible/.ssh/id_rsa): ansible
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ansible.
Your public key has been saved in ansible.pub.
The key fingerprint is:
SHA256:BqDEojHFrpggsChvnKvUGt3hKqLRV+zeSDkqvOWKgze ansible@ansible-server
The key's randomart image is:
+---[RSA 3072]-----+
|  +o .
|=.o. o
|+*. +
|B . o o
|=. . . = S
|EoB.o+=
|o*++ooo .
|+=+=.
|+=+o.
+---[SHA256]-----+
[ansible@ansible-server ~]$ █

[ansible@ansible-server ~]$ vim /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.14.144 docker.server
192.168.14.167 ansible.server
10.90.0.1 openshift.server
192.168.14.165 sonar.server
192.168.14.152 master.server
192.168.14.189 srv.accel.tan

~
~
```

Figure 5-26: Connection avec les autres serveurs

```
*
[ansible@ansible-server ~]$ ansible all -m ping
ansible.server | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
sonar.server | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
docker.server | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
```

Figure 5-27: Teste de L'installation et de la connection

D. Installation SonarQube

```
[sonarqube@soanr-server ~]$ sudo setenforce 0
[sudo] password for sonarqube:
[sonarqube@soanr-server ~]$ sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
[sonarqube@soanr-server ~]$
```

Figure 5-28: Désactivation de SELinux

```
[sonarqube@soanr-server ~]$ sudo vim /etc/sysctl.conf
# sysctl settings are defined through files in
# /usr/lib/sysctl.d/, /run/sysctl.d/, and /etc/sysctl.d/.
#
# Vendors settings live in /usr/lib/sysctl.d/.
# To override a whole file, create a new file with the same in
# /etc/sysctl.d/ and put new settings there. To override
# only specific settings, add a file with a lexically later
# name in /etc/sysctl.d/ and put new settings there.
#
# For more information, see sysctl.conf(5) and sysctl.d(5).

vm.max_map_count=262144
fs.file-max=65536
#ulimit -n 65536
#ulimit -u 4096

[sonarqube@soanr-server ~]$ sudo sysctl --system
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
kernel.yama.pttrace_scope = 0
* Applying /usr/lib/sysctl.d/50-coredump.conf ...
kernel.core_pattern = |/usr/lib/systemd/systemd-coredump %P %u %g %s %t %c %h %e
kernel.core_pipe_limit = 16
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1
kernel.kptr_restrict = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.promote_secondaries = 1
net.core.default_qdisc = fq_codel
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
* Applying /usr/lib/sysctl.d/50-libkapi-optmem_max.conf ...
net.core.optmem_max = 81920
* Applying /usr/lib/sysctl.d/50-pid-max.conf ...
kernel.pid_max = 4194304
* Applying /usr/lib/sysctl.d/60-libvirt.conf ...
fs.aio-max-nr = 1048576
* Applying /usr/lib/sysctl.d/60-qemu-postcopy-migration.conf ...
* Applying /etc/sysctl.d/99-sonarqube.conf ...
vm.max_map_count = 262144
fs.file-max = 65536
* Applying /etc/sysctl.d/99-sysctl.conf ...
vm.max_map_count = 262144
fs.file-max = 65536
* Applying /etc/sysctl.conf ...
vm.max_map_count = 262144
fs.file-max = 65536
[sonarqube@soanr-server ~]$
```

Figure 5-29: Configuration du sysctl

```
[sonarqube@soanr-server ~]$ sudo useradd sonar
[sonarqube@soanr-server ~]$ sudo dnf -y install https://download.postgresql.org/pub/repos/yum/repos/EL-8-x86_64/pgdg-redhat-repo-latest.noarch.rpm
Updating Subscription Management repositories.
Last metadata expiration check: 2:29:32 ago on Thu 22 Dec 2022 02:05:48 AM EST.
pgdg-redhat-repo-latest.noarch.rpm                               6.7 kB/s | 13 kB      00:01
Package pgdg-redhat-repo-42.0-28.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[sonarqube@soanr-server ~]$ sudo dnf -qy module disable postgresql
[sonarqube@soanr-server ~]$ sudo dnf -y install postgresql13 postgresql13-server
Updating Subscription Management repositories.
Last metadata expiration check: 2:30:38 ago on Thu 22 Dec 2022 02:05:48 AM EST.
Package postgresql13-13.9-1PGDG.rhel8.x86_64 is already installed.
Package postgresql13-server-13.9-1PGDG.rhel8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

Figure 5-30: Création de l'utilisateur et Installation des paquets de Postgress

```
[sonarqube@soanr-server ~]$ systemctl status postgresql-13
● postgresql-13.service - PostgreSQL 13 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-13.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2022-11-24 04:14:21 EST; 4 weeks 0 days ago
     Docs: https://www.postgresql.org/docs/13/static/
   Process: 3068 ExecStartPre=/usr/pgsql-13/bin/postgresql-13-check-db-dir ${PGDATA} (code=exited, status=0/SUCCESS)
    Main PID: 3074 (postmaster)
      Tasks: 12 (limit: 17676)
     Memory: 42.8M
    CGroup: /system.slice/postgresql-13.service
            └─ 3074 /usr/pgsql-13/bin/postmaster -D /var/lib/pgsql/13/data/
               └─ 3075 postgres: logger
                  └─ 3077 postgres: checkpointer
                     └─ 3078 postgres: background writer
                        └─ 3079 postgres: walwriter
                           └─ 3080 postgres: autovacuum launcher
                              └─ 3081 postgres: stats collector
                                 └─ 3082 postgres: logical replication launcher
                                    └─ 453308 postgres: sonar sonar_db 192.168.14.165(49792) idle
                                       └─ 453318 postgres: sonar sonar_db 192.168.14.165(35720) idle
                                          └─ 453647 postgres: sonar sonar_db 192.168.14.165(43542) idle
                                             └─ 453742 postgres: sonar sonar_db 192.168.14.165(46004) idle

Nov 24 04:14:20 soanr-server systemd[1]: Starting PostgreSQL 13 database server...
Nov 24 04:14:21 soanr-server postmaster[3074]: 2022-11-24 04:14:21.001 EST [3074] LOG:  redirecting log output to logging co
Nov 24 04:14:21 soanr-server postmaster[3074]: 2022-11-24 04:14:21.001 EST [3074] HINT:  Future log output will appear in dir
Nov 24 04:14:21 soanr-server systemd[1]: Started PostgreSQL 13 database server.
lines 1-26/26 (END)
```

Figure 5-31: Vérification du service de Postgress

MISE EN ŒUVRE

```
[sonarqube@soanr-server ~]$ sudo vim /var/lib/pgsql/13/data/postgresql.conf
# off, meaning append to existing files
# in all cases.
log_rotation_age = 1d          # Automatic rotation of logfiles will
log_rotation_size = 0         # happen after that time. 0 disables.
                                # Automatic rotation of logfiles will
                                # happen after that much log output.
                                # 0 disables.

log_line_prefix = '%m [%p] '  # special values:
                                #   %a = application name
                                #   %u = user name

log_timezone = 'America/New_York'

datestyle = 'iso, mdy'
#intervalstyle = 'postgres'
timezone = 'America/New_York'
#timezone_abbreviations = 'Default'   # Select the set of available time zone
lc_messages = 'en_US.UTF-8'         # locale for system error message
# strings
lc_monetary = 'en_US.UTF-8'         # locale for monetary formatting
lc_numeric = 'en_US.UTF-8'         # locale for number formatting
lc_time = 'en_US.UTF-8'            # locale for time formatting

# default configuration for text search
default_text_search_config = 'pg_catalog.english'

listen_addresses = '192.168.14.165'
```

Figure 5-32: Connection avec le serveur host

MISE EN ŒUVRE

```
[postgres@soanr-server ~]$ psql -U sonar -h 192.168.14.165 -p 5432 sonar_db
Password for user sonar:
psql (13.9)
Type "help" for help.

sonar_db=> \dt
          List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | active_rule_parameters | table | sonar
public | active_rules | table | sonar
public | alm_pats | table | sonar
public | alm_settings | table | sonar
public | analysis_properties | table | sonar
public | app_branch_project_branch | table | sonar
public | app_projects | table | sonar
public | audits | table | sonar
public | ce_activity | table | sonar
public | ce_queue | table | sonar
public | ce_scanner_context | table | sonar
```

Figure 5-34: Test de connexion avec la base de données

```
[sonarqube@soanr-server opt]$ sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.1.0.47736.zip
[sudo] password for sonarqube:
--2022-12-22 05:02:47-- https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.1.0.47736.zip
Resolving binaries.sonarsource.com (binaries.sonarsource.com) ... 99.86.91.19, 99.86.91.118, 99.86.91.83, ...
Connecting to binaries.sonarsource.com (binaries.sonarsource.com)|99.86.91.19|:443 ... connected.
HTTP request sent, awaiting response... 200 OK
Length: 309346924 (295M) [application/zip]
Saving to: 'sonarqube-9.1.0.47736.zip'

sonarqube-9.1.0.47736.zip      100%[=====] 295.02M  2.11MB/s   in 2m 19s
2022-12-22 05:05:06 (2.13 MB/s) - 'sonarqube-9.1.0.47736.zip' saved [309346924/309346924]

[sonarqube@soanr-server opt]$
[sonarqube@soanr-server opt]$ sudo chown -R sonar:sonar /opt/sonarqube
[sonarqube@soanr-server opt]$ sudo vim /opt/sonarqube/conf/wrapper.conf
# Path to JVM executable. By default it must be available in PATH.
# Can be an absolute path, for example:
#wrapper.java.command=/usr/bin/java
wrapper.java.command=/usr/bin/java

#
# DO NOT EDIT THE FOLLOWING SECTIONS
#

*****
# Wrapper Java
*****
wrapper.java.additional.1=-Dsonar.wrapped=true
wrapper.java.additional.2=-Djava.awt.headless=true
# extra args needed by hazelcast
wrapper.java.additional.3=--add-exports=java.base/jdk.internal.ref=ALL-UNNAMED
wrapper.java.additional.4=--add-opens=java.base/java.lang=ALL-UNNAMED
wrapper.java.additional.5=--add-opens=java.base/java.nio=ALL-UNNAMED
wrapper.java.additional.6=--add-opens=java.base/sun.nio.ch=ALL-UNNAMED
wrapper.java.additional.7=--add-opens=java.management/sun.management=ALL-UNNAMED
wrapper.java.additional.8=--add-opens=jdk.management/com.sun.management.internal=ALL-UNNAMED

wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperSimpleApp
wrapper.java.classpath.1=../lib/sonar-application-9.1.0.47736.jar
wrapper.java.classpath.2=../lib/jsw/wrapper-3.2.3.jar
wrapper.java.classpath.3=../lib/sonar-shutdowner-9.1.0.47736.jar
wrapper.java.library.path.1=/lib
wrapper.app.parameter.1=org.sonar.application.App
wrapper.java.initmemory=8
wrapper.java.maxmemory=32
```

Figure 5-33: Téléchargement et Intégration de Sonarqube avec Java


```
[sonarqube@soanr-server opt]$ sudo vim /opt/sonarqube/conf/sonar.properties
sonar.jdbc.username=sonar
sonar.jdbc.password=0
sonar.jdbc.url=jdbc:postgresql://192.168.14.165/sonar_db
sonar.web.javaOpts=-Xmx512m -Xms128m -XX:+HeapDumpOnOutOfMemoryError
sonar.web.port=9000

sonar.search.javaOpts=-Xmx512m -Xms128m -XX:MaxDirectMemorySize=256m -XX:+HeapDumpOnOutOfMemoryError
sonar.path.data=data
sonar.path.temp=temp

# Telemetry - Share anonymous SonarQube statistics
# By sharing anonymous SonarQube statistics, you help us understand how SonarQube is used so we can improve the product to work
# even better for you.
# We don't collect source code or IP addresses. And we don't share the data with anyone else.
# To see an example of the data shared: login as a global administrator, call the WS api/system/info and check the Statistics
# field.
#sonar.telemetry.enable=true
~
~
```

Figure 5-35: Intégration de Sonarqube avec Postgress

```
[sonarqube@soanr-server opt]$ sudo systemctl status sonarqube.service
● sonarqube.service - SonarQube service
   Loaded: loaded (/etc/systemd/system/sonarqube.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2022-11-24 04:17:21 EST; 4 weeks 0 days ago
     Process: 3333 ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop (code=killed, signal=TERM)
     Process: 4194 ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start (code=exited, status=0/SUCCESS)
    Main PID: 4243 (wrapper)
      Tasks: 197 (limit: 17676)
     Memory: 1.3G
      CGroup: /system.slice/sonarqube.service
              └─4243 /opt/sonarqube/bin/linux-x86-64/./wrapper /opt/sonarqube/bin/linux-x86-64/./../conf/wrapper.conf wrapper.s
                  └─4245 /usr/bin/java -Dsonar.wrapped=true -Djava.awt.headless=true --add-exports=java.base/jdk.internal.ref=ALL-UN
                  └─4276 /usr/lib/jvm/java-11-openjdk-11.0.17.0.8-2.el8_6.x86_64/bin/java -XX:+UseG1GC -Djava.io.tmpdir=/opt/sonarq
                  └─4403 /usr/lib/jvm/java-11-openjdk-11.0.17.0.8-2.el8_6.x86_64/bin/java -Djava.awt.headless=true -Dfile.encoding=U
                  └─4464 /usr/lib/jvm/java-11-openjdk-11.0.17.0.8-2.el8_6.x86_64/bin/java -Djava.awt.headless=true -Dfile.encoding=U

Nov 24 04:17:20 soanr-server systemd[1]: Starting SonarQube service ...
Nov 24 04:17:20 soanr-server sonar.sh[4194]: Starting SonarQube ...
Nov 24 04:17:21 soanr-server sonar.sh[4194]: Started SonarQube.
Nov 24 04:17:21 soanr-server systemd[1]: Started SonarQube service.
```

Figure 5-36: Vérification du service de Sonarqube

```
[sonarqube@soanr-server opt]$ sudo firewall-cmd --permanent --add-port=9000/tcp && sudo firewall-cmd --reload
Warning: ALREADY_ENABLED: 9000:tcp
success
success
[sonarqube@soanr-server opt]$
```

Figure 5-37: Accès au Firewall

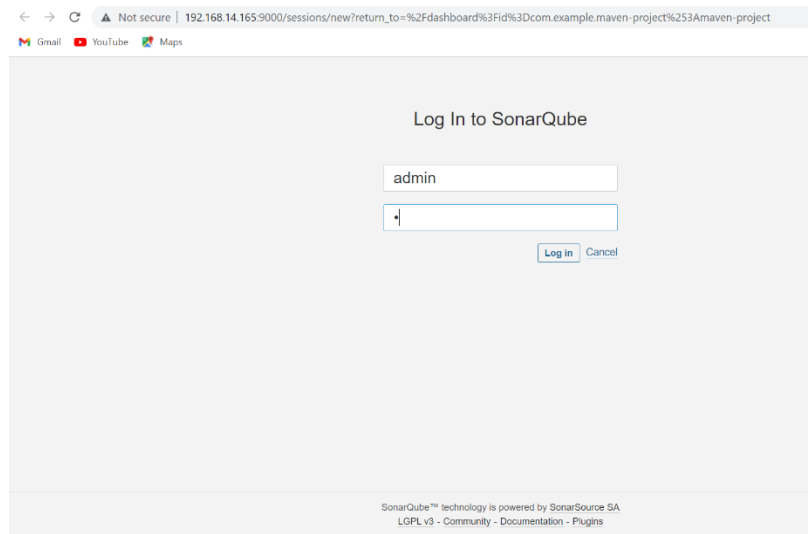


Figure 5-38: Interface Connexion à Sonarqube

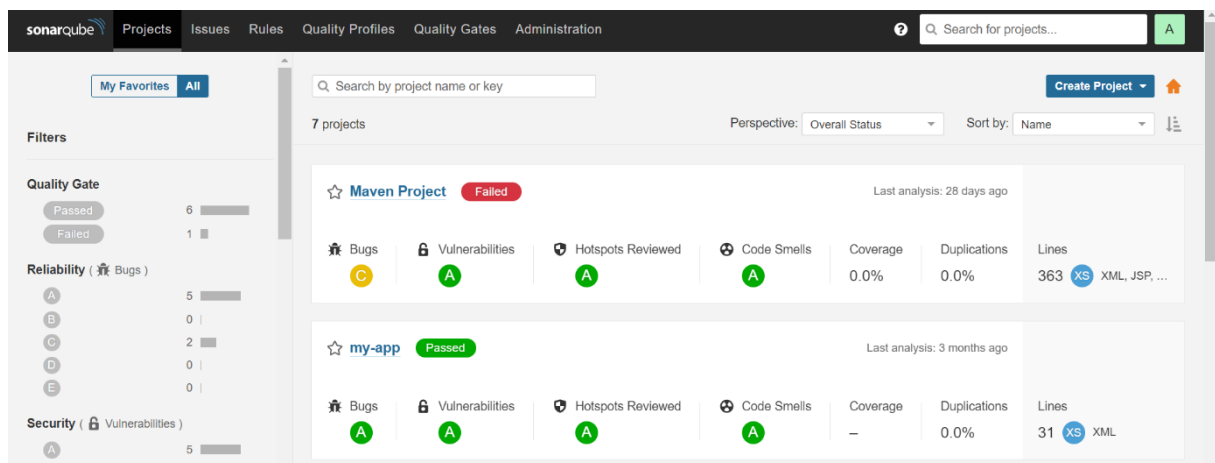


Figure 5-39: Interface Graphique de Sonarqube

CONCLUSION

CONCLUSION

Tout au long de notre étude nous avons vu les avantages et les inconvénients des méthodes de développements. La philosophie culturelle qui permet de mettre en place un modèle axé sur DevOps. Nous avons aussi parler de de l'importance de DevOps dans les entreprises et quelques pratiques clés de DevOps. Le modèle DevOps repose sur des outils efficaces pour aider les équipes à déployer et innover rapidement et efficacement pour leurs clients. Ces outils automatisent les tâches manuelles, aident les équipes à gérer des environnements complexes a différentes échelles et permettent aux ingénieurs de garder le contrôle sur la grande vitesse que le DevOps permet d'atteindre. Bien évidemment chaque projet DevOps est unique, surtout qu'il s'agit plus d'adopter une philosophie plutôt qu'une méthodologie. Cependant, la démarche doit être accompagnée, documentée et outillée, ce qui représente un coût qui doit être maîtrisé. Enfin la sécurité est donc une problématique à la fois pour les équipes d'exploitation et de développement. A ce titre, elle a beaucoup à gagner à s'inscrire dans les initiatives DevOps actuellement en plein essor.

Nous pouvons dire que Devops tend plus à la gestion des risques. Nous ouvrons comme perspective de mettre en place un environnement permettant d'avoir une vue d'ensemble de la qualité de service. Dans cette optique SLA¹⁴ tend à devenir un outil essentiel aux clients souhaitant bénéficier d'une sécurité infaillible sur certains de leurs niveaux de sécurité de stockage ainsi que sur la gestion de leurs données à caractère personnel. De nombreux indicateurs doivent être définis, analysés et contrôlés afin que la performance proposée par le prestataire soit maximisée.

¹⁴ Le service-level agreement ou « accord de niveau de service » est un document qui définit la qualité de service, prestation prescrite entre un fournisseur de service et un client. Autrement dit, il s'agit de clauses basées sur un contrat définissant les objectifs précis attendus et le niveau de service que souhaite obtenir un client de la part du prestataire et fixe les responsabilités.

BIBLIOGRAPHIE ET WEBOGRAPHIE

- Alliance, T. A. (s.d.). Récupéré sur <http://agilemanifesto.org/iso/fr/manifesto.html>
- Alliance, T. A. (s.d.). *Principes sous-jacents au manifeste*. Récupéré sur <http://agilemanifesto.org/iso/fr/principles.html>
- Amaury. (s.d.). *Le cycle en V*. Récupéré sur <https://www.geek-directeur-technique.com/2009/02/04/le-cycle-en-v>
- Felsen, N. (2017). *Effective DevOps with AWS*. Addison-Wesley.
- Genin, R. (s.d.). *Agile Tour Nantes 2011*. Récupéré sur <https://fr.slideshare.net/atnantes/agile-tour-nantes-2011-rmy-gnin-retours-dexpriece-sur-4-ans-dagilit-chez-orange>
- Getting Started with Pipelines*. (2021, 01 03). Récupéré sur <https://www.jenkins.io/pipeline/getting-started-pipelines/>
- Gonzalez, D. (2017). *Implementing Modern DevOps*. Addison-Wesley.
- Gunawardhana, S. (2021, 02 05). *Send Slack Notification from Jenkins*. Récupéré sur <https://medium.com/faun/send-slack-notifications-from-jenkins-1393fca282e6>
- Hat, I. R. (s.d.). Récupéré sur <https://www.redhat.com/fr/topics/security/container-security>
- Hat, I. R. (s.d.). *Kubernetes, qu'est-ce que c'est ?* Récupéré sur <https://www.redhat.com/fr/topics/containers/what-is-kubernetes>.
- Inc, S. (2015). *Docker Datasheet*. Récupéré sur https://cdn2.hubspot.net/hubfs/1958393/PDF/%20Docker_Datasheet.pdf.
- Inc., S. (In : (2015),, p. 1-4. doi). *"Docker Datasheet"*. Récupéré sur https://cdn2.hubspot.net/hubfs/1958393/PDF/%20Docker_Datasheet.pdf.
- IO, J. (2021, 02 05). *Pipeline Maven Integration Plugin*. Récupéré sur <https://www.jenkins.io/doc/pipeline/steps/pipeline-maven>
- Jenkins. (2020, 12 22). *Jenkins Redhat Packages*. Récupéré sur <https://pkg.jenkins.io/redhat-stable/>.
- Kumar, P. (2022, 3 9). *How to Install Kubernetes (k8s) Cluster on RHEL 8*. Récupéré sur <https://www.linuxtechi.com/how-to-install-kubernetes-cluster-rhel/>
- Leszko, R. (2017). *Continuous Delivery with Docker and Jenkins*. Addison-Wesley.
- MACVITTIE, D. (s.d.). *Atomic Chart Maps Out Its DevOps Ecosysteme*. Récupéré sur <https://devops.com/atomic-charts-maps-devops-ecosystem/>

Mutai, J. (2021, 10 18). *install-sonarqube-on-rocky-linux-centos*. Récupéré sur <https://computingforgeeks.com/install-sonarqube-on-rocky-linux-centos/>

Paradigm, V. (s.d.). *What is Scrum's Three Pillars ?* Récupéré sur <https://www.visual-paradigm.com/scrum/what-are-scrum-three-pillars/>

Projet, G. d. (s.d.). *Fonctionnement de Scrum*. Récupéré sur Processus Scrum: <https://gestiondeprojet.pm/gestion-de-projet-agile-avec-scrum/>

Shiferaw, A. G. (2021, 01 12). *Intro to Jenkins Pipelines and Publishing Over SSH*. Récupéré sur <https://dzone.com/articles/intro-to-jenkins-pipeline-and-using-publish-over-s>

Systems, C. (s.d.). *Waterfall Product Development*. Récupéré sur <https://castellansystems.com/kb/Waterfall.cshtml>.

Unknown. (s.d.). Récupéré sur <https://app.emaze.com/@AOCCZRIFR#1>.