

Automatisation et orchestration du déploiement d'Applications

Réalisé par :

KAMMOUN HALE

ALOUÏ ARBIA

BORGI ALA

CII2-SSIR-A

Année Universitaire : 2023-2024

Table de matières

<i>Introduction Générale</i>	<i>1</i>
<i>Chapitre 1 : Contexte générale de projet</i>	<i>2</i>
Introduction	2
1. Présentation de l'entreprise d'accueil	2
1.1. Les services offerts par Tekup	2
1.2. Qualités des études.....	2
1.3. Diverses méthodes de financement.....	3
1.4. Certifications internationales	3
2. Présentation du cadre du projet	3
2.1. Problématique	3
2.2. Etude et critiques de l'existant	4
3. Objectif du projet	5
4. Étude des besoins	5
4.1. Besoins fonctionnels	6
4.2. Besoins non fonctionnels	6
5. Planning du projet	8
Conclusion	8
<i>Chapitre 2 : Etat de l'art</i>	<i>9</i>
Introduction	9
1. Déploiement automatisé :	9
2. Intégration continue (CI) :	9
3. Déploiement continu (CD) :	9
4. Pipeline CI/CD :	9
5. Scripting et automatisation :	11

6. Comparaison des solutions et outils de DevOps :	12
Conclusion	15
<i>Chapitre 3 : Réalisation</i>	<i>16</i>
Introduction	16
1. Architecture de la solution	16
2. Étapes de réalisation :	18
2.1. Installation et configuration des instances EC2 :	18
2.2. Mise en œuvre des outils DevOps :	19
2.2.1. Intégration entre Github et jenkins :	19
2.2.2. Écriture de Dockerfile & Push du code vers GitHub :	20
2.2.3. Création et Configuration du Pipeline :	21
Conclusion	33
<i>Conclusion générale</i>	<i>34</i>
<i>Webographie</i>	<i>35</i>

Liste des figures

FIGURE 1: CI/CD PIPELINE	10
FIGURE 2: ARCHITECTURE DE LA SOLUTION PROPOSEE	16
FIGURE 3: VERIFICATION DE L'INSTALLATION DE ANSIBLE	18
FIGURE 4: VERIFICATION DE L'INSTALLATION DE JENKINS ET CONFIRMATION DU SERVICE	18
FIGURE 5: VERIFICATION DE L'INSTALLATION DE MINIKUBE ET DOCKER	19
FIGURE 6: RÉALISATION DE WEBHOOK	19
FIGURE 7: REALISATION DE PUSH VERS GITHUB	20
FIGURE 8: REALISATION DE PREMIER STAGE SUR JENKINS.	21
FIGURE 9: LANCEMENT DE « BUILD » POUR LE PREMIER STAGE.	21
FIGURE 10: VERIFICATION DE PREMIER STAGE.....	22
FIGURE 11: AUTOMATISATION DE BUILD DES STAGES.	23
FIGURE 12: CONFIGURATION DE DEUXIEME STAGE.	23
FIGURE 13: EXECUTION DE DEUXIEME STAGE.	24
FIGURE 14: VERIFICATION DE DEUXIEME STAGE.	24
FIGURE 15: CONFIGURATION DE TROISIEME STAGE.....	25
FIGURE 16: LANCEMENT DE TROISIEME STAGE AVEC SUCCES	25
FIGURE 17: CONFIGURATION DE QUATRIEME STAGE.....	26
FIGURE 18: LANCEMENT DE QUATRIEME BUILD.....	26
FIGURE 19: VERIFICATION DE QUATRIEME STAGE.....	27
FIGURE 21: : VERIFICATION DES IMAGES DANS DOCKERHUB.....	28
FIGURE 20: CONFIGURATION ET LANCEMENT DE CINQUIEME STAGE.	28
FIGURE 22: CONTENU DE PLAYBOOK DEPLOYMENT.YML	29
FIGURE 23:CONTENU DE PLAYBOOK SERVICE.YML.....	29
FIGURE 24:CONTENU DE PLAYBOOK ANSIBLE.YML	30
FIGURE 25: CONFIGURATION DE SSH ENTRE K8S ET ANSIBLE.	30
FIGURE 26: VERIFICATION DE CONNECTIVITE VERS KUBERNETES.	31
FIGURE 27: CONFIGURATION DE SIXIEME STAGE.	31
FIGURE 28: EXECUTION DE SIXIEME STAGE.	32
FIGURE 29: CONFIGURATION DE SEPTIEME STAGE.	32
FIGURE 30: EXECUTION DE SEPTIEME STAGE.	33

Liste des tableaux

TABLE 1: PLANNING D'EXÉCUTION	8
TABLE 2: COMPARAISON DES OUTILS DEVOPS.	12

Introduction Générale

La transformation numérique redéfinit comment nous développons et déployons les logiciels.

Les équipes informatiques doivent réagir rapidement aux besoins des utilisateurs tout en garantissant la fiabilité et la sécurité des systèmes, pour relever ces défis, beaucoup se tournent vers les pratiques DevOps, qui encouragent la collaboration et l'automatisation des processus.

Le déploiement automatisé des applications, élément clé du DevOps, accélère la mise sur le marché tout en réduisant les risques, en automatisant les tâches répétitives, les équipes peuvent améliorer l'efficacité opérationnelle et répondre plus rapidement aux besoins des utilisateurs.

Ce rapport explore en détail cette approche, mettant en lumière ses avantages, ses défis et les meilleures pratiques pour une mise en œuvre réussie.

En résumé, il vise à fournir une ressource précieuse pour les professionnels de l'informatique intéressés par le DevOps et le déploiement automatisé des applications, afin de promouvoir l'innovation et l'excellence opérationnelle dans leurs organisations.

Chapitre 1 : Contexte générale de projet

Introduction

Avant de débiter notre projet, ce premier chapitre établit le contexte de notre étude. Nous aborderons ensuite la problématique du projet, la comparaison des outils existants, ainsi que la solution que nous avons choisie, ainsi que les objectifs à atteindre.

1. Présentation de l'entreprise d'accueil

TEK-UP, l'École Privée Supérieure de Technologie et d'Ingénierie, est une institution qui offre la possibilité de bénéficier d'une éducation académique capable de doter les étudiants des connaissances, des compétences et du savoir-faire nécessaires pour devenir un ingénieur réussi dans un proche avenir.

L'université est accréditée par le Ministère de l'Enseignement Supérieur, de la Recherche Scientifique et des Technologies de l'Information et de la Communication depuis août 2014. TEK-UP décerne le Diplôme National en Ingénierie Informatique ou Télécommunications.

Leur mission est d'éduquer les ingénieurs et de les coacher pour innover et développer de nouvelles idées tout en intégrant la technologie dans leurs projets et leur programme. Une compréhension approfondie du rôle de l'ingénierie sera développée, permettant à nos étudiants d'identifier les besoins du monde d'aujourd'hui et de concevoir les meilleures solutions technologiques [1].

1.1. Les services offerts par Tekup

Tekup propose à ses clients plusieurs services qui sont décrits dans ce qui suit.

1.2. Qualités des études

TEK-UP offre un environnement d'apprentissage de haute qualité. L'éducation à l'université couvre des compétences spécifiques et des méthodes scientifiques qui permettront aux diplômés d'améliorer leurs qualifications.

1.3. Diverses méthodes de financement

Il existe de nombreuses façons de financer vos études à TEK-UP. Pour vous aider à vous offrir une éducation adaptée à votre situation financière, TEK-UP peut vous accorder une méthode de financement spécifique où vous pouvez payer vos études après l'obtention de votre diplôme une fois que vous avez trouvé un emploi.

1.4. Certifications internationales

Chaque étudiant à l'opportunité de passer près de 12 certifications internationales telles que CCNA, CCNP, LPIC-1, LPIC-2, Microsoft MCP, Oracle JAVA, etc.

2. Présentation du cadre du projet

Dans cette section, nous présentons la problématique du projet tout en analysant l'état existant.

2.1. Problématique

Le déploiement manuel des applications présente plusieurs inconvénients majeurs. Tout d'abord, il nécessite une intervention humaine pour exécuter des tâches répétitives telles que la création et la configuration des environnements, le déploiement du code et la vérification de la cohérence des configurations, cette approche est non seulement chronophage, mais elle est également sujette à des erreurs humaines, ce qui peut entraîner des versions défectueuses, des conflits de configuration et des temps d'arrêt imprévus.

Lent et sujet aux erreurs :

Les processus manuels impliquent une intervention humaine à chaque étape du déploiement des applications. Cela signifie que chaque action, de la configuration de l'environnement à la vérification de la cohérence des configurations, est sujette à des retards dus au temps nécessaire pour effectuer les tâches et aux possibilités d'erreurs humaines.

Par exemple, une mauvaise configuration ou un oubli lors du déploiement manuel peut entraîner des problèmes de compatibilité, des pannes ou des dysfonctionnements des applications. En conséquence, le processus global de déploiement prend plus de temps et peut entraîner des retards dans la mise à disposition des nouvelles fonctionnalités pour les utilisateurs.

Retards et indisponibilité :

Les déploiements manuels sont souvent lents, ce qui peut entraîner des retards dans la mise à disposition des nouvelles fonctionnalités ou des correctifs critiques pour les utilisateurs finaux. De plus, la lenteur des déploiements augmente le risque d'indisponibilité des applications pendant la phase de déploiement, ce qui peut impacter négativement l'expérience utilisateur et même entraîner des pertes financières pour l'entreprise. Par exemple, si une mise à jour doit être déployée rapidement pour corriger une vulnérabilité de sécurité, les retards dus aux processus manuels peuvent laisser l'application exposée à des attaques potentielles, ce qui met en danger la sécurité des données et la réputation de l'entreprise.

Coûts élevés :

La gestion manuelle des déploiements nécessite des ressources importantes en termes de main-d'œuvre et de temps. Les équipes doivent consacrer des heures à effectuer les tâches de déploiement, ce qui peut entraîner une augmentation des coûts de main-d'œuvre.

De plus, les erreurs humaines peuvent entraîner des coûts supplémentaires liés aux temps d'arrêt, aux réparations et aux retours en arrière pour corriger les problèmes causés par les déploiements manuels. En automatisant ces processus, les entreprises peuvent réduire les coûts de main-d'œuvre, minimiser les erreurs humaines et optimiser l'utilisation des ressources, ce qui entraîne une réduction globale des coûts de maintenance.

2.2. Etude et critiques de l'existant

Pour réussir notre projet, il est crucial de comparer les solutions utilisées, disponibles sur le marché. Dans le domaine du développement logiciel, il existe une variété de méthodes et d'approches pour le déploiement des applications, chacune avec ses propres avantages et inconvénients. Les méthodes traditionnelles de déploiement manuel impliquent généralement une série d'étapes réalisées par des opérateurs humains, depuis la configuration des serveurs jusqu'à la mise en production des applications. Bien que largement utilisées par le passé, ces méthodes manuelles sont de plus en plus considérées comme inefficaces dans un environnement de développement moderne.

Cependant, au fil des ans, de nouvelles approches ont émergé pour répondre à ces défis. Parmi celles-ci, les pratiques DevOps et l'automatisation du déploiement des applications ont gagné en popularité en raison de leur capacité à accélérer les cycles de développement, à

améliorer la qualité des logiciels et à réduire les coûts opérationnels. Les équipes DevOps adoptent une approche intégrée du développement et de l'exploitation, favorisant la collaboration, l'automatisation et l'amélioration continue des processus.

En outre, de nombreuses entreprises ont adopté ces pratiques et ces outils avec succès, ce qui témoigne de leur efficacité et de leur pertinence dans un large éventail de contextes industriels. Des études de cas et des exemples concrets montrent comment l'automatisation du déploiement des applications a permis d'améliorer l'efficacité opérationnelle, de réduire les délais de mise sur le marché et d'augmenter la qualité des logiciels. afin de choisir celle qui répondra le mieux à nos besoins.

Renforcer la sécurité des déploiements :

En intégrant des pratiques de sécurité dès le processus de développement, nous visons à renforcer la sécurité des déploiements. L'automatisation des tests de sécurité et la gestion sécurisée des secrets avec Ansible et Vault contribueront à réduire les risques de failles de sécurité.

3. Objectif du projet

L'objectif de notre projet est de mettre en place une infrastructure DevOps robuste et automatisée pour faciliter le développement, le déploiement et la gestion des applications. Plus précisément, nous visons à atteindre les objectifs suivants :

Optimiser l'utilisation des ressources :

En utilisant des conteneurs Docker et une orchestration avec Kubernetes, notre objectif est d'optimiser l'utilisation des ressources informatiques.

Nous pourrions ainsi scaler dynamiquement les applications en fonction de la charge, maximisant ainsi l'efficacité opérationnelle et réduisant les coûts.

4. Étude des besoins

L'étude des besoins dans notre projet de déploiement automatisé des applications DevOps est cruciale pour garantir que l'architecture mise en place réponde efficacement aux exigences de l'organisation. Voici les principaux aspects abordés dans cette étude :

4.1. Besoins fonctionnels

Automatisation du processus de déploiement :

Il est essentiel que le processus de déploiement des applications soit entièrement automatisé, depuis la compilation du code jusqu'à son déploiement sur l'infrastructure cible.

Gestion des environnements :

Nous devons être en mesure de gérer efficacement les environnements de développement, de test et de production, en assurant la cohérence des configurations et des versions déployées.

Intégration continue et déploiement continu (CI/CD) :

Nous avons besoin d'une intégration continue fluide et de déploiements continus fiables pour garantir des mises à jour régulières et sans heurts des applications.

Scalabilité et haute disponibilité :

L'architecture doit être conçue pour être scalable et hautement disponible, afin de pouvoir répondre aux fluctuations de la charge et garantir une disponibilité optimale des applications.

Sécurité :

La sécurité des déploiements et des données est une priorité. Nous devons mettre en place des mécanismes de sécurité robustes pour protéger les applications contre les menaces et les vulnérabilités.

Suivi et monitoring :

Un système de suivi et de monitoring complet est nécessaire pour surveiller les performances des applications, détecter les problèmes et les erreurs, et réagir rapidement en cas d'incidents.

4.2. Besoins non fonctionnels

Facilité d'utilisation :

Les outils et les processus doivent être intuitifs et faciles à utiliser pour l'ensemble des équipes impliquées, des développeurs aux opérationnels.

Performance :

L'architecture doit être optimisée pour garantir des performances élevées et des temps de réponse rapides, même dans des environnements à haute charge.

Flexibilité :

Il est important que l'architecture soit flexible et extensible, capable de s'adapter aux évolutions futures des besoins de l'organisation.

Coût :

L'optimisation des coûts est un facteur important. Nous devons choisir des solutions qui offrent un bon rapport qualité-prix tout en répondant à nos besoins opérationnels.

5. Planning du projet

La planification est la clé principale de la réussite d'un projet. En effet, le planning aide bien subdiviser le travail et séparer les tâches à réaliser, il offre une meilleure estimation et gestion de temps nécessaire pour chaque tâche. L'objectif principal de la planification des tâches est de fournir une vision du projet et de son déroulement. Elle permet de diriger la réalisation d'une étape et la durée qui lui a déjà été consacrée, ainsi que l'avancement du projet dans son ensemble, avec répercussions de chaque tâche sur la date de fin du projet.

Le tableau 2 montre le planning que nous avons adapté pour mener à bien notre réalisation des différentes parties du projet.

Table 1: Planning d'exécution

Étape	Tâche	Date de réalisation
0. Installation et configuration des ec2	Installation et configuration de ec2 Jenkins	Terminé
	Installation et configuration de ec2 Ansible	Terminé
	Installation et configuration de ec2 k8s	Terminé
1. Écriture du Dockerfile	Le développeur crée un fichier Dockerfile spécifiant les dépendances, la configuration et les étapes nécessaires pour construire une image Docker de l'application.	Terminé
2. Push du Dockerfile vers GitHub	Le développeur pousse le Dockerfile ainsi que tout le code source de l'application vers un référentiel GitHub.	Terminé
3. Notification à Jenkins via Webhook	Un webhook est configuré dans GitHub pour notifier Jenkins chaque fois qu'un nouveau code est poussé (commit ou push)	Terminé
4. Build par Jenkins	Jenkins reçoit la notification et déclenche un processus de build.	Terminé
5. SSH vers Ansible	Jenkins récupère le code source depuis le référentiel GitHub.	Terminé
	Jenkins établit une connexion SSH vers le serveur Ansible.	Terminé
6. Déploiement et Tagging par Ansible	Ansible utilise le Dockerfile pour construire une image Docker.	Terminé
	Ansible effectue un tagging de l'image.	Terminé
	Ansible pousse l'image tagged vers Docker Hub.	Terminé
7. SSH vers Kubernetes Cluster	Ansible établit une connexion SSH avec le serveur du cluster Kubernetes.	Terminé
8. Mise à jour de l'application sur Kubernetes	Ansible exécute un <u>playbook</u> Kubernetes qui utilise la commande <u>kubectl</u> pour mettre à jour l'image de l'application sur le cluster.	Terminé
	Le <u>playbook</u> peut tirer la dernière image depuis Docker Hub ou construire une nouvelle image à partir du Dockerfile, puis la déployer sur le cluster.	29/4/2024
9. Accessibilité via IP et <u>Port</u> :	Une fois le conteneur déployé sur le cluster Kubernetes, l'application est accessible via une adresse IP et <u>un port spécifiés</u> dans le service Kubernetes.	29/4/2024

Conclusion

Nous avons consacré ce premier chapitre à l'introduction du cadre général du projet à travers une présentation du contexte. Par la suite, nous avons étudié les différentes technologies candidates pour la réalisation du projet et cela à travers une étude comparative entre elles.

Chapitre 2 : Etat de l'art

Introduction

Dans ce chapitre, nous exposons l'impératif d'automatiser le déploiement. Pour ce faire, nous commençons par définir quelques concepts de base qui éclairent notre compréhension du projet.

1. Déploiement automatisé :

Le déploiement automatisé fait référence au processus de déploiement d'applications et de services de manière automatisée, sans intervention manuelle significative. Il vise à réduire les erreurs, à accélérer les déploiements et à améliorer la cohérence des environnements de production.

2. Intégration continue (CI) :

L'intégration continue est une pratique de développement logiciel dans laquelle les développeurs fusionnent régulièrement leur code dans un référentiel partagé. Chaque fusion déclenche des builds automatisés et des tests pour vérifier l'intégrité du code.

3. Déploiement continu (CD) :

Le déploiement continu étend le concept d'intégration continue en automatisant également le déploiement du code vers l'environnement de production ou de test. Il vise à rendre les déploiements plus rapides, plus sûrs et plus fiables.

4. Pipeline CI/CD :

Un pipeline CI/CD est une série d'étapes automatisées qui permettent de construire, de tester et de déployer une application de manière cohérente et reproductible. Il peut inclure des phases telles que la compilation du code, les tests unitaires, les tests d'intégration, la création d'images Docker et le déploiement sur un cluster Kubernetes.

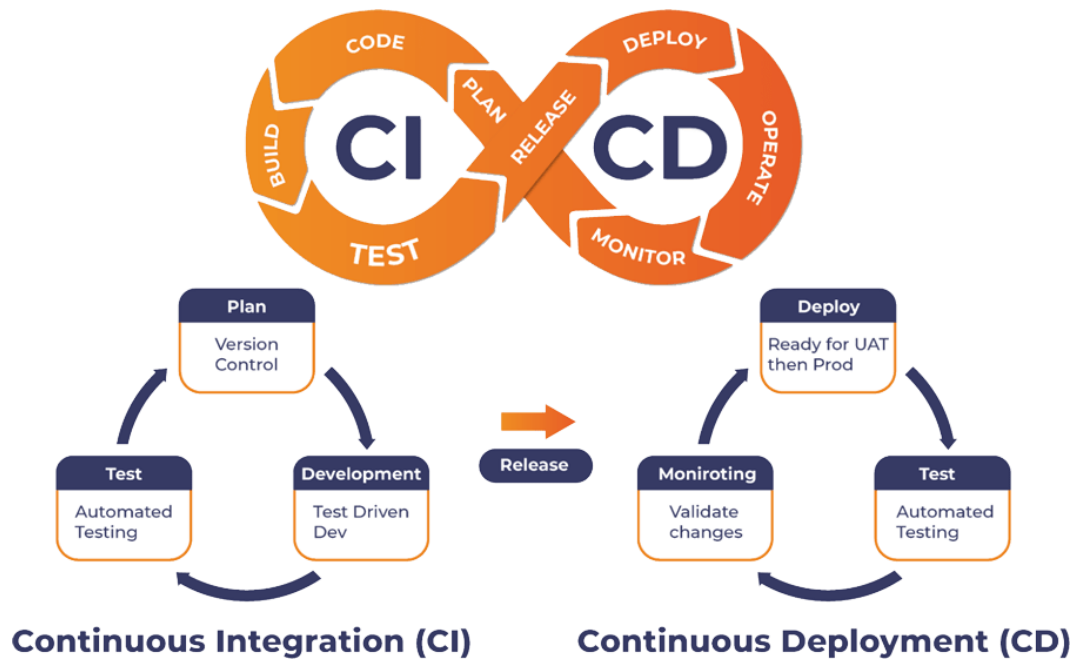


Figure 1: CI/CD Pipeline

La figure 2 précédente présente un diagramme qui illustre les étapes clés du pipeline d'intégration continue et de livraison continue (CI/CD) pour le développement et le déploiement d'applications logicielles. Il met l'accent sur la nature automatisée et itérative du processus, garantissant une livraison de logiciels efficace et fiable, ces étapes clés sont les suivantes :

Planification :

Cette étape consiste à définir les objectifs de développement, les tâches et le calendrier du projet. Elle établit la direction et la portée globales du processus de développement logiciel.

Codage :

Les développeurs écrivent le code de l'application, en respectant les normes de codage et les bonnes pratiques. Cela comprend la mise en œuvre de fonctionnalités, la correction de bogues et l'amélioration des fonctionnalités de l'application.

Tests :

Des outils de test automatisés sont utilisés pour évaluer en permanence la qualité et la fonctionnalité du code. Ces tests identifient et signalent toute erreur ou défaut, garantissant que le code répond aux spécifications souhaitées.

Construction :

Le code est compilé et assemblé en un package déployable. Ce processus implique l'intégration de différents composants de l'application et sa préparation pour le déploiement.

Déploiement :

Le package déployable est automatiquement publié sur l'environnement cible, tel qu'un serveur de staging ou de production. Cela rend la version mise à jour de l'application accessible aux utilisateurs.

Surveillance :

L'application déployée est continuellement surveillée en matière de performances, de stabilité et de sécurité. Cette approche proactive permet d'identifier et de résoudre les problèmes potentiels avant qu'ils n'impactent les utilisateurs.

Retour d'information :

Le feedback est recueilli auprès des utilisateurs, des testeurs et d'autres parties prenantes pour évaluer les performances de l'application et identifier les points à améliorer. Ce feedback est intégré aux futurs cycles de développement.

Dans l'ensemble, le CI/CD est devenu une pratique essentielle dans le développement de logiciels moderne, permettant aux équipes de fournir des applications de haute qualité avec une plus grande agilité et efficacité.

5. Scripting et automatisation :

Le Scripting et l'automatisation consistent à écrire des scripts ou des configurations qui automatisent les tâches répétitives et manuelles associées au déploiement et à la gestion des applications. Cela peut inclure des scripts pour la configuration des serveurs, le déploiement d'applications et la mise à jour des environnements.

En comprenant ces concepts de base, nous sommes en mesure d'apprécier pleinement l'importance de l'automatisation du déploiement dans notre projet DevOps. En automatisant les processus de build, de test et de déploiement, nous visons à accélérer les cycles de développement, à réduire les erreurs humaines et à garantir des déploiements plus fiables et cohérents. Ce chapitre jettera les bases nécessaires pour explorer en profondeur la mise en œuvre de l'automatisation du déploiement dans les sections suivantes du rapport.

6. Comparaison des solutions et outils de DevOps :

Table 2: Comparaison des outils DevOps.

Outil	Avantages	Inconvénients
Git	- Versionnement efficace du code - Gestion flexible des branches - Large adoption et support communautaire	- Courbe d'apprentissage initiale pour les utilisateurs novices - Besoin d'une bonne gestion des conflits de fusion
Subversion	- Facilité d'utilisation pour les utilisateurs novices - Support centralisé et robuste	- Performance moindre pour les grands dépôts - Moins d'options pour la gestion des branches par rapport à Git
GitHub	- Plateforme de partage de code largement utilisée - Fonctionnalités sociales et de collaboration avancées	- Certaines fonctionnalités avancées sont limitées aux comptes premium - Dépendance à un service tiers pour l'hébergement
GitLab	- Solution complète avec intégration continue et déploiement continu intégrés - Possibilité d'hébergement en interne	- Courbe d'apprentissage initiale pour les utilisateurs novices - Certaines fonctionnalités peuvent être complexes à configurer
Bitbucket	- Intégration transparente avec d'autres produits Atlassian comme Jira et Trello - Option pour un hébergement en interne	- Moins de fonctionnalités sociales et de collaboration par rapport à GitHub - Peut sembler moins intuitif pour certains utilisateurs
Jenkins	- Hautement personnalisable avec une grande variété de plugins disponibles - Forte communauté d'utilisateurs	- Interface utilisateur datée et moins intuitive que certaines alternatives - Gestion complexe des configurations pour les pipelines
Ansible	- Facile à apprendre et à mettre en œuvre - Aucun agent requis sur les nœuds cibles	- Moins adapté à des scénarios complexes que certains outils de gestion de configuration comme Puppet ou Chef

Puppet	- Gestion centralisée de la configuration - Déclaration de l'état désiré pour garantir la cohérence de la configuration	- Courbe d'apprentissage initiale plus longue que certains outils comme Ansible - Configuration initiale peut être complexe
Chef	- Scalabilité et gestion centralisée - Flexibilité pour gérer des configurations complexes	- Courbe d'apprentissage initiale plus longue que certains outils comme Ansible - Configuration peut être complexe
Docker	- Isolation efficace des applications - Portabilité entre les environnements de développement, de test et de production	- Configuration réseau parfois complexe - Gestion de la persistance des données pour les conteneurs
Kubernetes	- Orchestration des conteneurs pour le déploiement et la gestion à grande échelle - Scalabilité et haute disponibilité	- Courbe d'apprentissage abrupte pour les nouveaux utilisateurs - Gestion complexe des ressources et des configurations
Terraform	- Provisioning d'infrastructure multi-cloud - Déclaration de l'infrastructure comme code	- Courbe d'apprentissage initiale pour les utilisateurs novices - Complexité accrue pour les infrastructures complexes
Grafana	- Interface utilisateur élégante et conviviale - Prise en charge de diverses sources de données et de visualisations	- Configuration initiale peut être complexe pour les nouveaux utilisateurs - Moins d'options de personnalisation par rapport à certaines alternatives
Prometheus	- Métriques puissantes et collecte de données - Facilité d'intégration avec d'autres outils DevOps	- Courbe d'apprentissage initiale pour les utilisateurs novices - Gestion des alertes peut nécessiter une configuration minutieuse
ELK Stack	- Analyse de logs performante - Visualisation avancée des	- Configuration initiale peut être complexe pour les nouveaux

	données - Évolutivité et flexibilité	utilisateurs - Gestion des pipelines peut être complexe
--	--------------------------------------	---

À la suite de l'analyse comparative de toutes les solutions, Après une analyse comparative approfondie de toutes les solutions disponibles, notre choix s'est porté sur Jenkins, Ansible, Kubernetes, GitHub et Docker pour plusieurs raisons clés.

Jenkins :

Jenkins est choisi pour son haut degré de personnalisation et sa grande variété de plugins disponibles, ce qui nous permet de répondre efficacement à nos besoins spécifiques en matière d'intégration continue et de déploiement continu. Sa forte communauté d'utilisateurs et son historique de fiabilité en font un choix solide pour automatiser nos pipelines de développement.

Ansible :

Ansible a été retenu en raison de sa facilité d'apprentissage et de sa mise en œuvre, ainsi que de son approche agentless, ce qui élimine le besoin d'installer des agents sur les nœuds cibles. Sa capacité à gérer la configuration des serveurs de manière simple et efficace correspond à nos besoins en matière d'automatisation des opérations système.

Kubernetes :

Kubernetes est notre choix pour l'orchestration des conteneurs en raison de sa capacité à gérer le déploiement et la gestion des applications à grande échelle de manière efficace et fiable. Sa scalabilité, sa haute disponibilité et sa flexibilité répondent à nos exigences croissantes en matière de déploiement d'applications conteneurisées dans des environnements complexes.

GitHub :

GitHub est sélectionné comme plateforme de gestion de code source en raison de sa large adoption, de ses fonctionnalités sociales et de collaboration avancées, ainsi que de son intégration transparente avec nos outils de développement existants. Sa robustesse et sa popularité en font un choix idéal pour héberger nos dépôts de code et faciliter la collaboration entre les membres de l'équipe.

Docker :

Docker est notre choix pour la conteneurisation des applications en raison de sa simplicité, de son efficacité et de sa portabilité. Sa capacité à isoler les applications avec tous leurs éléments nécessaires et à garantir une cohérence entre les environnements de développement, de test et de production répond à nos besoins en matière de déploiement d'applications dans des environnements hétérogènes.

En combinant ces outils, nous sommes en mesure de construire un pipeline DevOps robuste et efficace, depuis le développement et les tests jusqu'au déploiement et à la gestion des applications en production. Chaque outil a été sélectionné pour ses fonctionnalités spécifiques et sa capacité à répondre à nos besoins opérationnels, tout en contribuant à notre objectif global d'amélioration de la collaboration, de l'efficacité et de la fiabilité de nos processus de développement logiciel.

Conclusion

Dans ce chapitre, nous avons expliqué les notions théoriques et fondamentales des technologies utilisées pour réaliser notre solution proposée.

Chapitre 3 : Réalisation

Introduction

Dans ce chapitre dédié à la réalisation, nous abordons la phase concrète de notre projet. Nous plongerons dans les détails opérationnels, explorant les étapes spécifiques qui ont transformé notre vision en une réalité tangible.

1. Architecture de la solution

Cette figure présente les étapes clés de l'architecture supportée pour la publication d'une application web sur le web.

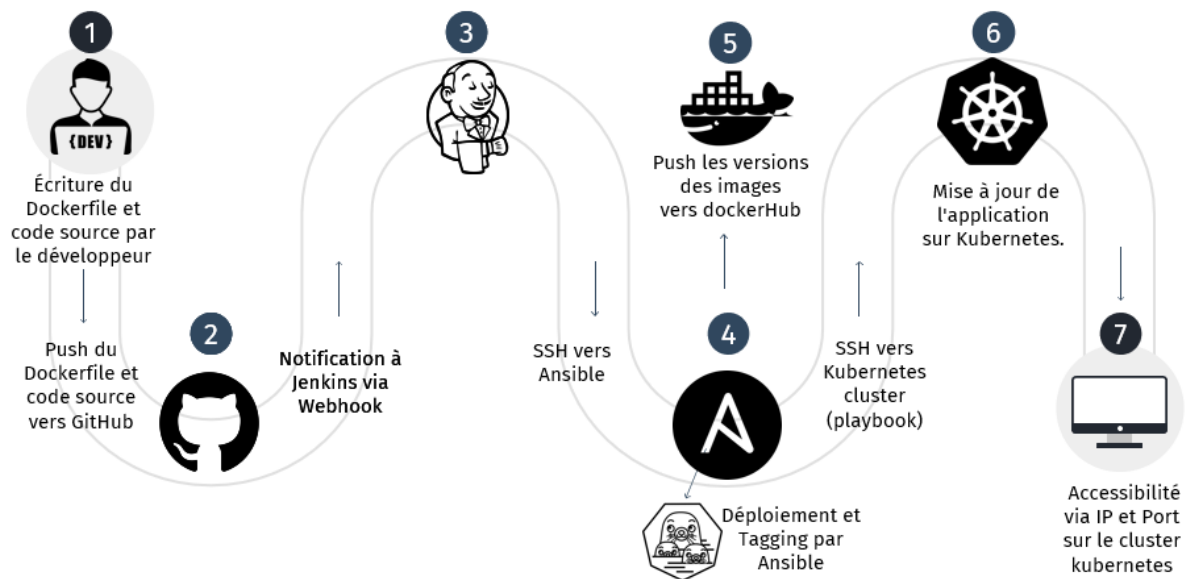


Figure 2: Architecture de la solution proposée

Notre projet vise à mettre en place une architecture de déploiement automatisé des applications, intégrant les meilleures pratiques DevOps pour garantir un développement logiciel efficace, rapide et fiable. Cette architecture repose sur l'utilisation synergique des technologies suivantes : Docker, GitHub, Jenkins, Ansible et Kubernetes.

L'objectif principal est de créer un pipeline de déploiement continu robuste, depuis l'écriture du code par les développeurs jusqu'à la mise en production des applications sur un cluster Kubernetes. Pour ce faire, nous avons conçu un processus fluide et automatisé, permettant une intégration continue, des tests automatisés et un déploiement sans interruption.

Le développement commence avec la rédaction d'un Dockerfile par les développeurs, décrivant les dépendances et les étapes de construction de l'image Docker de l'application. Une fois le code et le Dockerfile poussés vers GitHub, Jenkins est notifié via un webhook pour déclencher un processus de build.

Jenkins récupère le code source depuis GitHub, construit l'image Docker et la pousse vers Docker Hub. En parallèle, Jenkins établit une connexion SSH avec Ansible, qui orchestre le déploiement de l'application sur le cluster Kubernetes. Ansible utilise le Dockerfile pour construire l'image Docker et la déployer sur le cluster Kubernetes.

Une fois l'application déployée sur le cluster, elle est accessible via une adresse IP et un port spécifié dans le service Kubernetes, assurant ainsi une disponibilité continue pour les utilisateurs finaux.

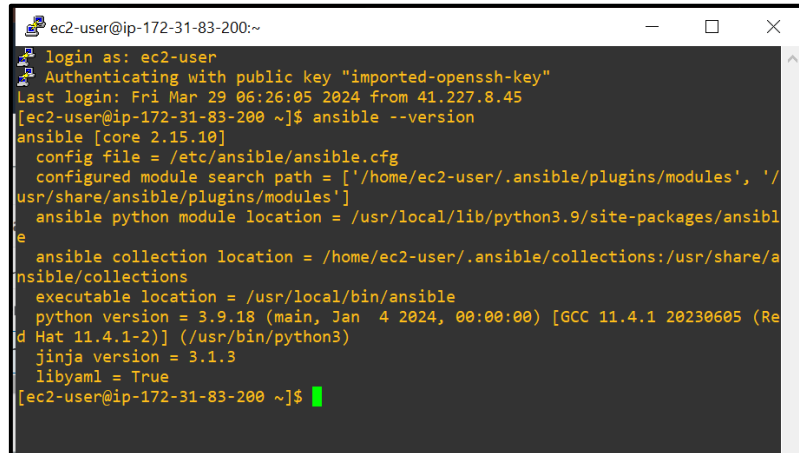
Cette architecture offre de nombreux avantages, notamment une automatisation complète du processus de déploiement, une meilleure gestion des environnements de développement et de production, une scalabilité et une fiabilité accrue, ainsi qu'une réduction significative des délais de déploiement et des erreurs humaines. En fin de compte, notre objectif est de fournir une solution DevOps hautement efficace et performante, répondant aux besoins évolutifs de notre organisation en matière de développement logiciel.

2. Étapes de réalisation :

Le projet sera réalisé en plusieurs étapes :

2.1.Installation et configuration des instances EC2 :

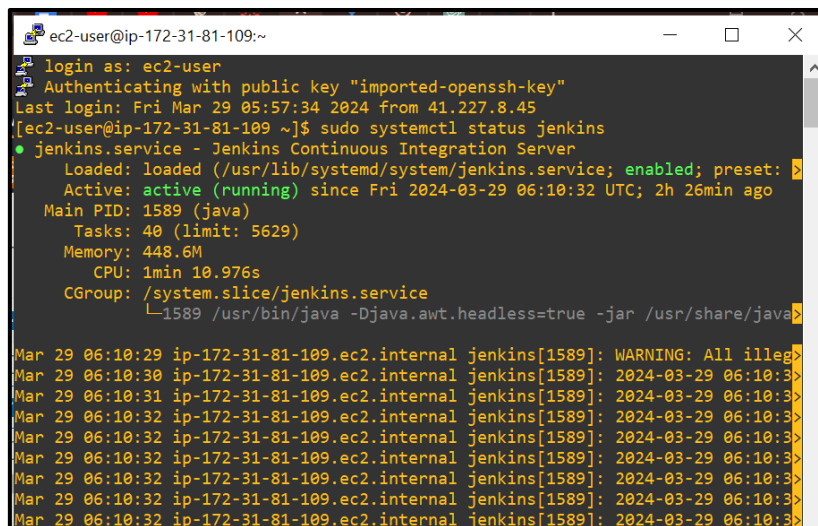
Nous avons créé trois instances EC2 : une pour Jenkins, une pour Ansible et une pour Kubernetes, ainsi on a utilisé PUTTY comme ssh agent.



```
ec2-user@ip-172-31-83-200:~  
login as: ec2-user  
Authenticating with public key "imported-openssh-key"  
Last login: Fri Mar 29 06:26:05 2024 from 41.227.8.45  
[ec2-user@ip-172-31-83-200 ~]$ ansible --version  
ansible [core 2.15.10]  
  config file = /etc/ansible/ansible.cfg  
  configured module search path = ['/home/ec2-user/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']  
  ansible python module location = /usr/local/lib/python3.9/site-packages/ansible  
  ansible collection location = /home/ec2-user/.ansible/collections:/usr/share/ansible/collections  
  executable location = /usr/local/bin/ansible  
  python version = 3.9.18 (main, Jan  4 2024, 00:00:00) [GCC 11.4.1 20230605 (Red Hat 11.4.1-2)] (/usr/bin/python3)  
  jinja version = 3.1.3  
  libyaml = True  
[ec2-user@ip-172-31-83-200 ~]$
```

Figure 3: Vérification de l'installation de ansible

La figure 3 démontre l'installation de ansible-core avec succès.



```
ec2-user@ip-172-31-81-109:~  
login as: ec2-user  
Authenticating with public key "imported-openssh-key"  
Last login: Fri Mar 29 05:57:34 2024 from 41.227.8.45  
[ec2-user@ip-172-31-81-109 ~]$ sudo systemctl status jenkins  
● jenkins.service - Jenkins Continuous Integration Server  
   Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: disabled)  
   Active: active (running) since Fri 2024-03-29 06:10:32 UTC; 2h 26min ago  
     Main PID: 1589 (java)  
       Tasks: 40 (limit: 5629)  
      Memory: 448.6M  
         CPU: 1min 10.976s  
    CGroup: /system.slice/jenkins.service  
            └─1589 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/...  
  
Mar 29 06:10:29 ip-172-31-81-109.ec2.internal jenkins[1589]: WARNING: All illegal  
Mar 29 06:10:30 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:30  
Mar 29 06:10:31 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:31  
Mar 29 06:10:32 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:32  
Mar 29 06:10:32 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:32  
Mar 29 06:10:32 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:32  
Mar 29 06:10:32 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:32  
Mar 29 06:10:32 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:32  
Mar 29 06:10:32 ip-172-31-81-109.ec2.internal jenkins[1589]: 2024-03-29 06:10:32
```

Figure 4: Vérification de l'installation de Jenkins et Confirmation du Service

La figure 4 démontre la confirmation de son installation réussie en tant que service.

```
ubuntu@ip-172-31-85-11: ~  
ubuntu@ip-172-31-85-11:~$ minikube start  
* minikube v1.32.0 on Ubuntu 22.04  
* Using the docker driver based on existing profile  
  
X Exiting due to PROVIDER_DOCKER_NEWGRP: "docker version --format <no value>:<no value>:<no value>" exit status 1: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/version": dial unix /var/run/docker.sock: connect: permission denied  
* Suggestion: Add your user to the 'docker' group: 'sudo usermod -aG docker $USER && newgrp docker'  
* Documentation: https://docs.docker.com/engine/install/linux-postinstall/  
  
ubuntu@ip-172-31-85-11:~$ docker --version  
Docker version 24.0.2, build cb74dfc  
ubuntu@ip-172-31-85-11:~$
```

Figure 5: vérification de l'installation de minikube et docker

La figure 5 démontre la version de minikube ainsi que docker d'où la confirmation de l'installation.

2.2.Mise en œuvre des outils DevOps :

Nous avons ensuite configuré les intégrations entre les outils :

2.2.1. Intégration entre Github et jenkins :

Nous avons configuré un webhook entre github et jenkins en précisant l'adresse ip public de l'instance jenkins, pour le notifier à chaque fois un développeur réalise un « push » dans le répertoire « devops_project »

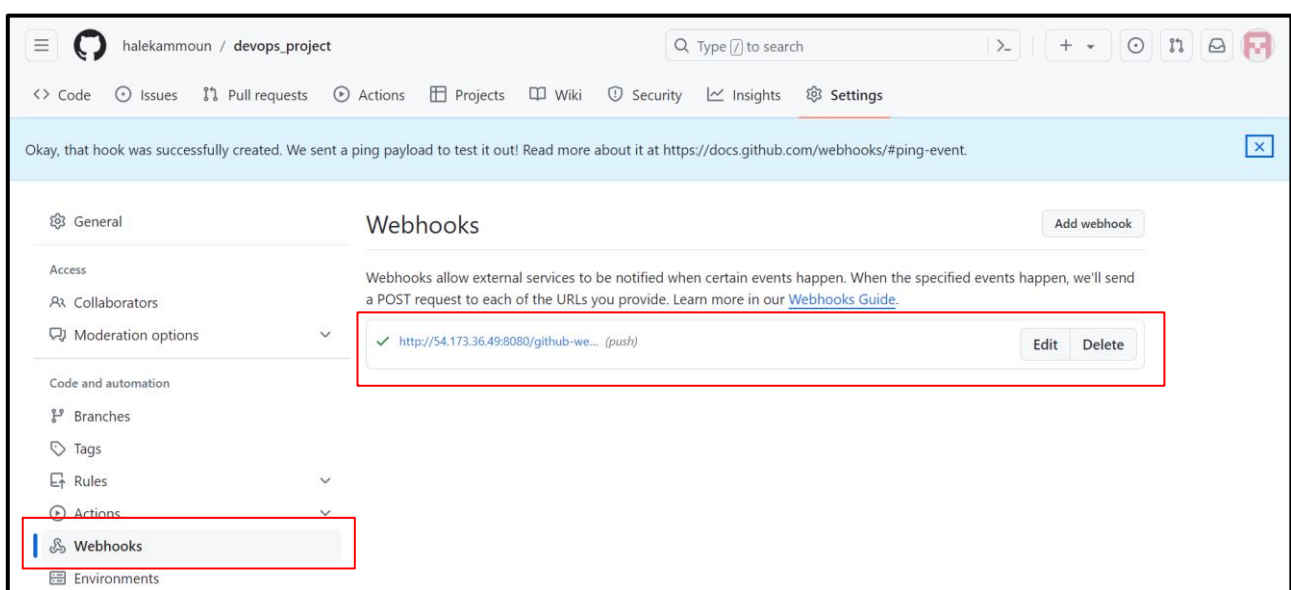


Figure 6: Réalisation de webhook

2.2.2. Écriture de Dockerfile & Push du code vers GitHub :

Le développeur crée un fichier Dockerfile qui définit l'environnement de développement et de construction de l'application.

Le Dockerfile spécifie les dépendances, les instructions de build et l'image Docker finale.

Ensuite, le développeur affecte un Push du code vers GitHub ainsi, jenkins sera notifié automatiquement grâce au webhook déjà configuré.

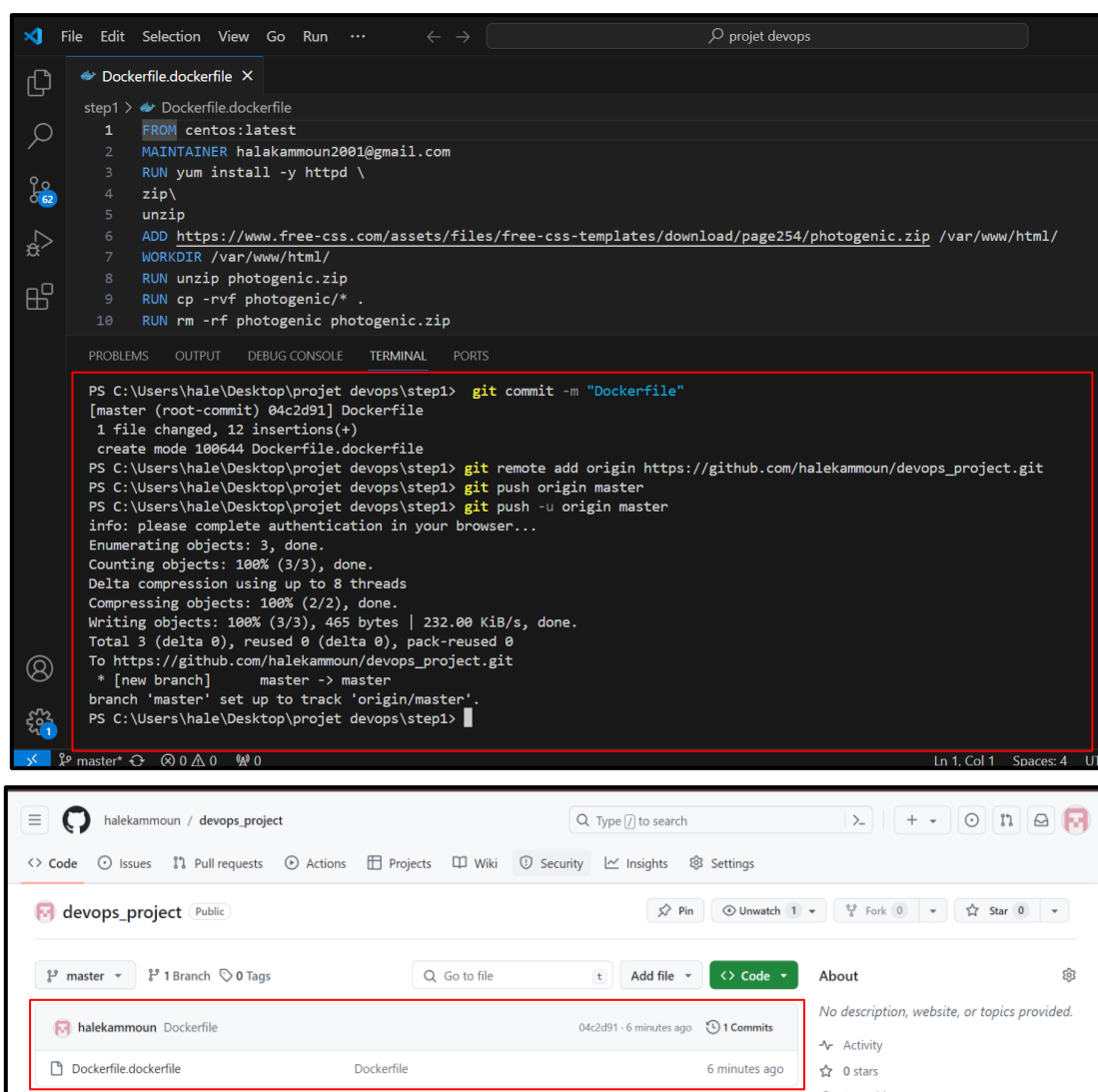


Figure 7: Réalisation de push vers Github

2.2.3. Création et Configuration du Pipeline :

Nous avons initié la création d'un pipeline nommé "our_first_pipeline", les étapes suivantes ont été effectuées :

Le premier stage « git checkout » :

pour accéder au répertoire contenant le Dockerfile sur GitHub come montre la figure 8.

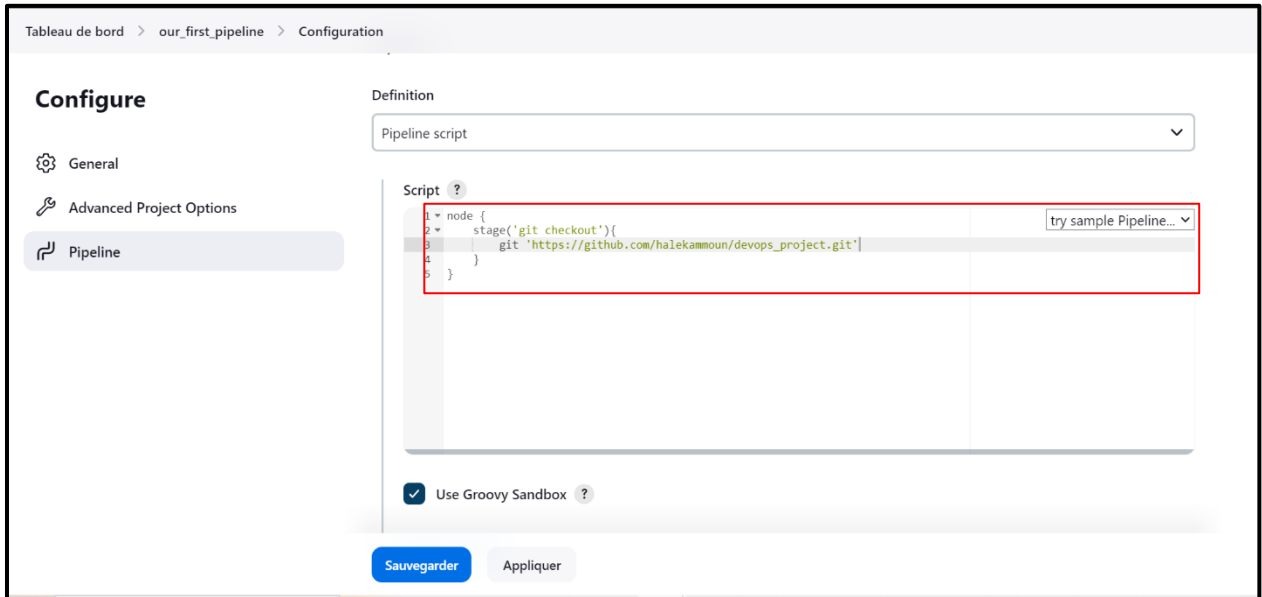


Figure 8: Réalisation de premier stage sur jenkins.

La figure 9 montre que le stage est lancé avec succès.

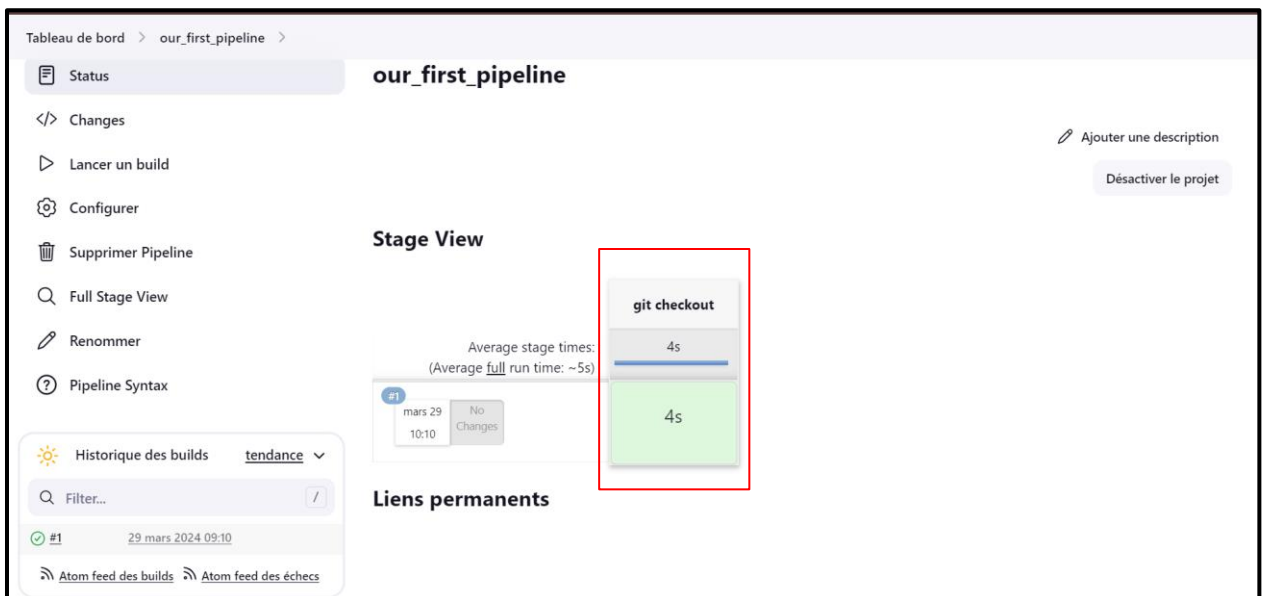


Figure 9: Lancement de « build » pour le premier stage.

Lorsqu'on a vérifié dans la machine nous avons remarqué que le répertoire existant dans le github est transféré avec succès dans les répertoires de Jenkins.

La figure 10 démontre que dockerfile a été transsféré à jenkins sous le chemin par défaut
« /var/lib/jenkins/workspace/our_first_pipeline/ »

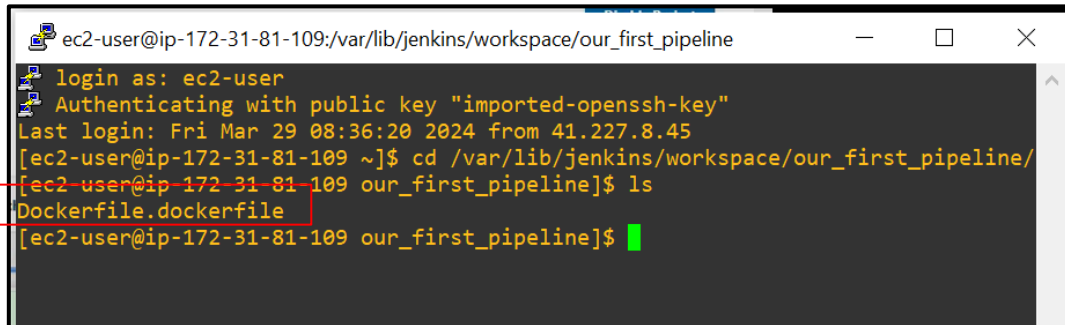
A terminal window titled 'ec2-user@ip-172-31-81-109:/var/lib/jenkins/workspace/our_first_pipeline' shows the following commands and output: 'login as: ec2-user', 'Authenticating with public key "imported-openssh-key"', 'Last login: Fri Mar 29 08:36:20 2024 from 41.227.8.45', 'cd /var/lib/jenkins/workspace/our_first_pipeline/', and 'ls'. The output of 'ls' is 'Dockerfile.dockerfile', which is highlighted with a red box. The prompt is '[ec2-user@ip-172-31-81-109 our_first_pipeline]\$'.

Figure 10: Vérification de premier stage.

Pour automatiser ce processus de build, nous avons effectué les actions suivantes :

Dans Jenkins :

Accès à la configuration du projet.

- Configuration des déclencheurs de build en sélectionnant "GitHub hook trigger for GITScm polling" en appliquant et sauvegardant les modifications.

Dans GitHub :

- Accès aux paramètres du répertoire.
- Ajout d'un webhook en spécifiant l'URL de Jenkins (<http://54.173.36.49:8080/github-webhook/>) et en choisissant le type JSON.
- Saisie du secret avec le jeton généré depuis Jenkins.
- Enregistrement et rafraîchissement de la page.

Pour confirmer que le build est automatisé, nous avons effectué les actions suivantes :

- Modification du Dockerfile
- Utilisation des commandes Git pour ajouter, commettre et pousser les modifications vers GitHub.
- Observation de l'automatisation du build dans l'historique des builds (#2) comme montre la figure 11 suivante.

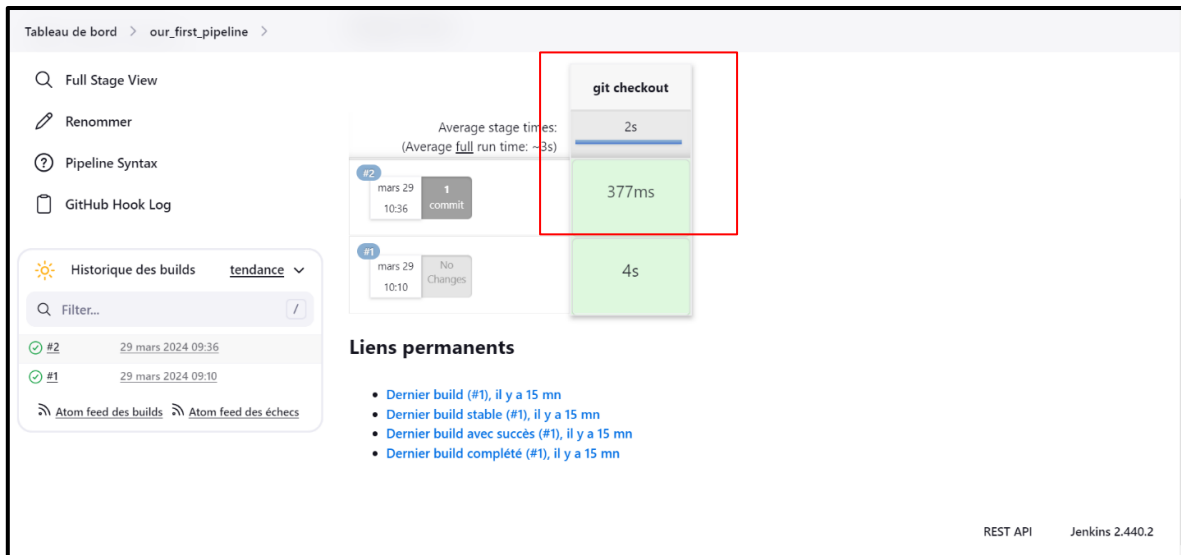


Figure 11: automatisation de build des stages.

Le deuxième stage « sending the dockerfile to ansible » :

Pour la transmission du Dockerfile de Jenkins à Ansible via SSH et pour automatiser l'envoi du Dockerfile de Jenkins à Ansible, nous avons utilisé SSH Agent pour permettre une connexion sécurisée sans mot de passe.

Voici les étapes suivies :

Ajout de la clé privée ansible.pem dans le pipeline Jenkins via l'interface Pipeline Syntax.

Intégration d'une étape de SSH dans le script du pipeline pour transférer le Dockerfile vers Ansible.

Enregistrement des modifications, puis exécution d'un nouveau build (#4).



Figure 12: Configuration de deuxième stage.

La figure 13 suivante montre que le stage est réalisé avec succès.



Figure 13: Exécution de deuxième stage.

Après le transfert du Dockerfile à Ansible, nous avons vérifié que le processus s'est déroulé avec succès :

Dans Ansible :

Connexion en tant que superutilisateur (sudo su).

Vérification du répertoire pour s'assurer que le Dockerfile a été supprimé (ls).

Reconstruction du Dockerfile en relançant le build (#5).

Vérification que le Dockerfile a été recréé (ls).

La figure si dessous montre le processus de vérification de deuxième stage.

```
root@ip-172-31-83-200:/home/ec2-user
[ec2-user@ip-172-31-83-200 ~]$ pwd
/home/ec2-user
[ec2-user@ip-172-31-83-200 ~]$ sudo su
[root@ip-172-31-83-200 ec2-user]# ls
ansible.sh Dockerfile.dockerfile
[root@ip-172-31-83-200 ec2-user]# rm Dockerfile.dockerfile
rm: remove regular file 'Dockerfile.dockerfile'? y
[root@ip-172-31-83-200 ec2-user]# ls
ansible.sh
[root@ip-172-31-83-200 ec2-user]# ls
ansible.sh
[root@ip-172-31-83-200 ec2-user]# ls
ansible.sh Dockerfile.dockerfile
[root@ip-172-31-83-200 ec2-user]#
```

Figure 14: Vérification de deuxième stage.

Cette étape garantit que le Dockerfile est correctement transmis à Ansible et que le processus de création fonctionne comme prévu.

Le troisième stage « Docker build the image »

C'est pour la construction d'image dans Ansible :

Afin de construire l'image dans Ansible, nous avons ajouté une troisième étape au pipeline Jenkins pour effectuer le SSH vers le serveur Ansible et construire l'image Docker.

Nous avons ajouté une nouvelle étape au script du pipeline Jenkins pour effectuer la connexion SSH vers le serveur Ansible et réaliser la construction de l'image Docker comme montre la figure 15.



Figure 15: configuration de troisième stage.

Puis nous avons enregistré des modifications et nous avons lancé le Build comme montre la figure 16 suivante.

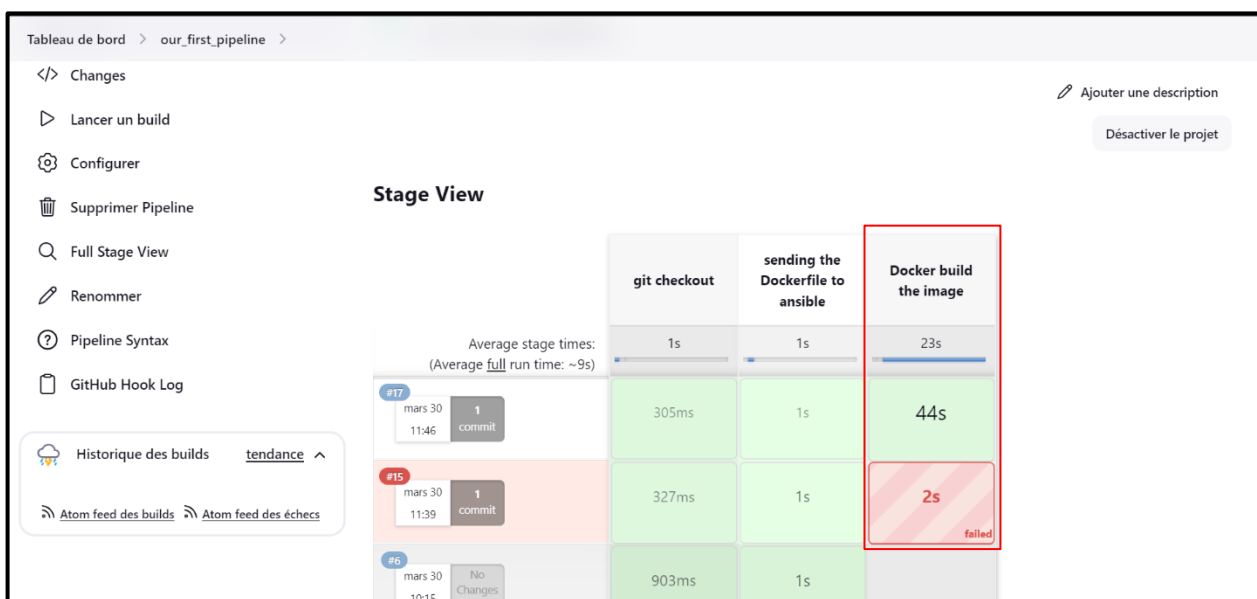


Figure 16: lancement de troisième stage avec succès

Le quatrième stage « Docker image tagging » :

Ce stage est pour tagger les images pour distinguer les versions des images et surtout de distinguer la dernière image «Latest ».

Cette étape assure que l'image Docker est construite dans l'environnement Ansible conformé, la figure 17 suivante montre la configuration nécessaire de ce stage.

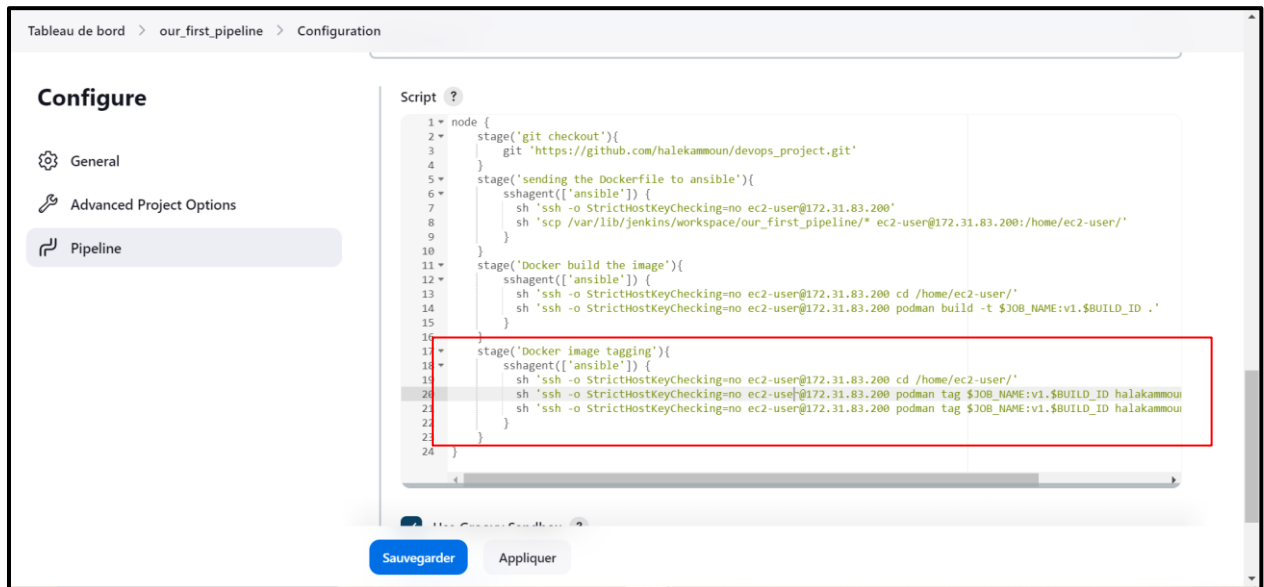


Figure 17: Configuration de quatrième stage.

Ensuite, nous avons enregistré des modifications et nous avons lancé le Build comme montre la figure 16 suivante.

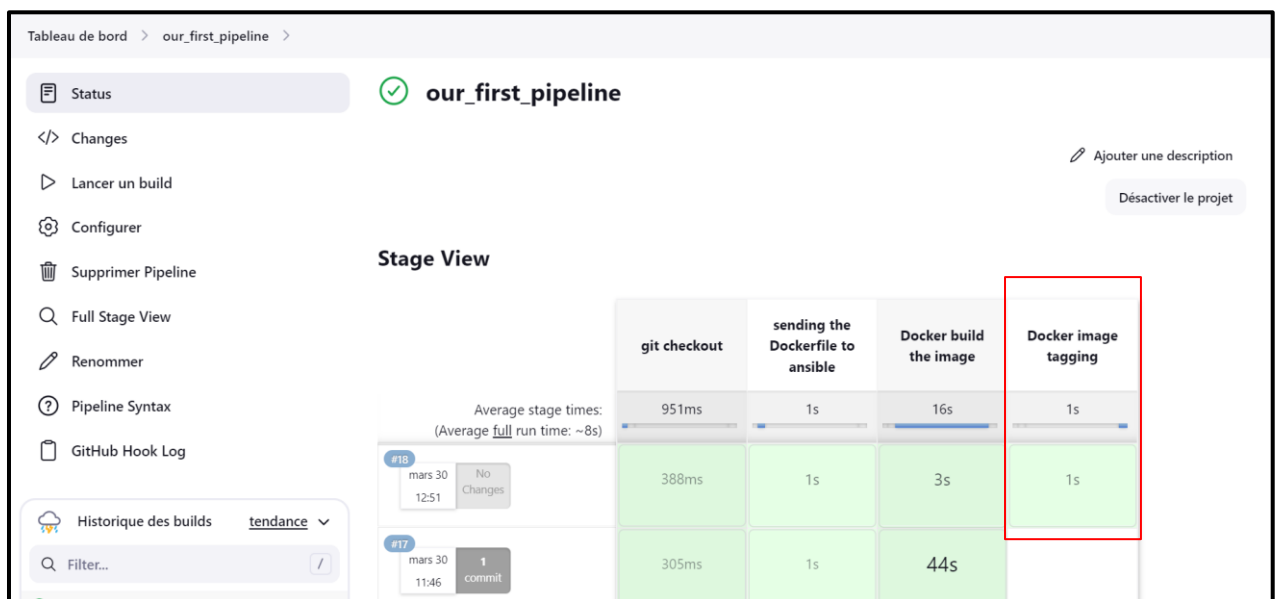
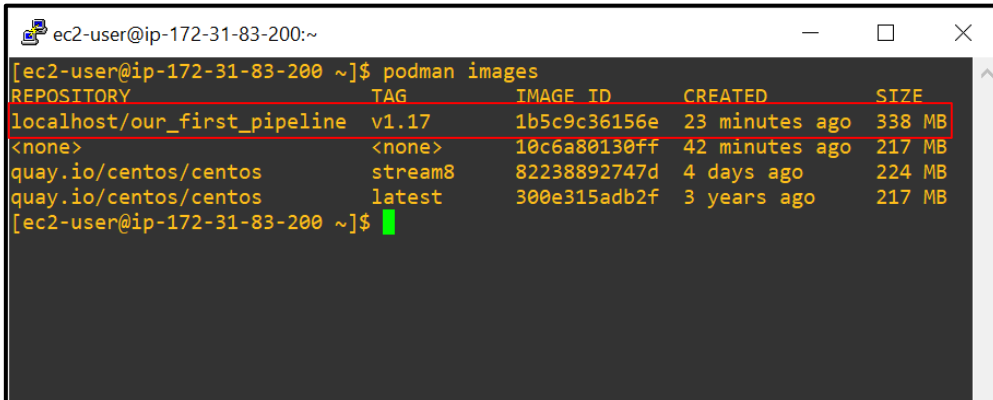


Figure 18: Lancement de quatrième build.

Pour vérifier que l'image a été construite avec succès et correctement taggée, nous avons effectué la commande « podman images » pour vérifier la présence de l'image Docker comme montre la figure suivante.



```
ec2-user@ip-172-31-83-200:~  
[ec2-user@ip-172-31-83-200 ~]$ podman images  
REPOSITORY          TAG          IMAGE ID       CREATED        SIZE  
localhost/our_first_pipeline v1.17       1b5c9c36156e  23 minutes ago 338 MB  
<none>              <none>      10c6a80130ff  42 minutes ago 217 MB  
quay.io/centos/centos      stream8      82238892747d  4 days ago    224 MB  
quay.io/centos/centos      latest       300e315adb2f  3 years ago    217 MB  
[ec2-user@ip-172-31-83-200 ~]$
```

Figure 19: Vérification de quatrième stage.

Cette étape assure que l'image Docker est construite dans l'environnement Ansible conformément aux exigences du projet, et qu'elle est correctement taggée pour une référence future.

Le cinquième stage « Push docker images to Dockerhub » :

est pour pousser les Images Taggées depuis Ansible vers Docker Hub comme montre la figure 20:

Pour publier les images taggées depuis Ansible vers Docker Hub, nous avons mis en place un stage supplémentaire dans le pipeline Jenkins pour effectuer les actions suivantes :

- SSH vers le serveur Ansible.
- Connexion à Docker Hub.
- Utilisation de Podman pour pousser l'image taggée vers Docker Hub.
- Connexion à Docker Hub :

Dans Ansible, nous nous sommes connectés à Docker Hub en utilisant la commande `podman login registry-1.docker.io`.

- Construction du Nouveau Stage :

Nous avons ajouté les commandes nécessaires pour effectuer la connexion et le push dans le pipeline Jenkins.

Puis nous avons enregistré des modifications et nous avons lancé le Build comme montre la figure 20 et 21 suivantes :

Pipeline script

```

19 sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 cd /home/ec2-user/'
20 sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 podman tag $JOB_NAME:v1.$BUILD_ID halakammoun'
21 sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 podman tag $JOB_NAME:v1.$BUILD_ID halakammoun'
22 }
23
24 stage('push docker images to dockerhub'){
25   sshagent(['ansible']) {
26     withCredentials([string(credentialsId: 'dockerhub_password', variable: 'dockerhub_passwd')]) {
27       sh "podman login -u halakammoun -p ${dockerhub_passwd} docker.io"
28       sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 podman push halakammoun/$JOB_NAME:v1.$BUILD_ID'
29       sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 podman push halakammoun/$JOB_NAME:latest'
30     }
31   }
32 }
33
34 }

```

Jenkins

rechercher (CTRL+K)

Tableau de bord > our_first_pipeline >

Status our_first_pipeline

Changes

Lancer un build

Configurer

Supprimer Pipeline

Full Stage View

Renommer

Pipeline Syntax

GitHub Hook Log

Historique des builds

Stage View

Ajouter une description

Désactiver le projet

Average stage times:
(Average full run time: ~8s)

	git checkout	sending the Dockerfile to ansible	Docker build the image	Docker image tagging	push docker images to dockerhub
Average	570ms	1s	3s	1s	3s
#29	276ms	1s	2s	1s	3s

Figure 21: Configuration et lancement de cinquième stage.

Après avoir poussé les images taggées vers Docker Hub, nous avons vérifié leur disponibilité et leur mise à jour sur Docker Hub ainsi, nous avons confirmé que l'image "latest" était correctement mise à jour avec la dernière version poussée depuis Ansible.

dockerhub

Explore Repositories Organizations

Search Docker Hub

halakammoun / Repositories / our_first_pipeline / Tags

Using 0 of 1 private repositories. [Get more](#)

General Tags Builds Collaborators Webhooks Settings

Sort by Newest Filter Tags Delete

TAG	Digest	OS/ARCH	Last pull	Compressed Size
latest	b3e36bb25bd	linux/amd64	a minute ago	156.07 MB
v1.28	b3e36bb25bd	linux/amd64	a minute ago	156.07 MB

Figure 20: : Vérification des images dans DockerHub

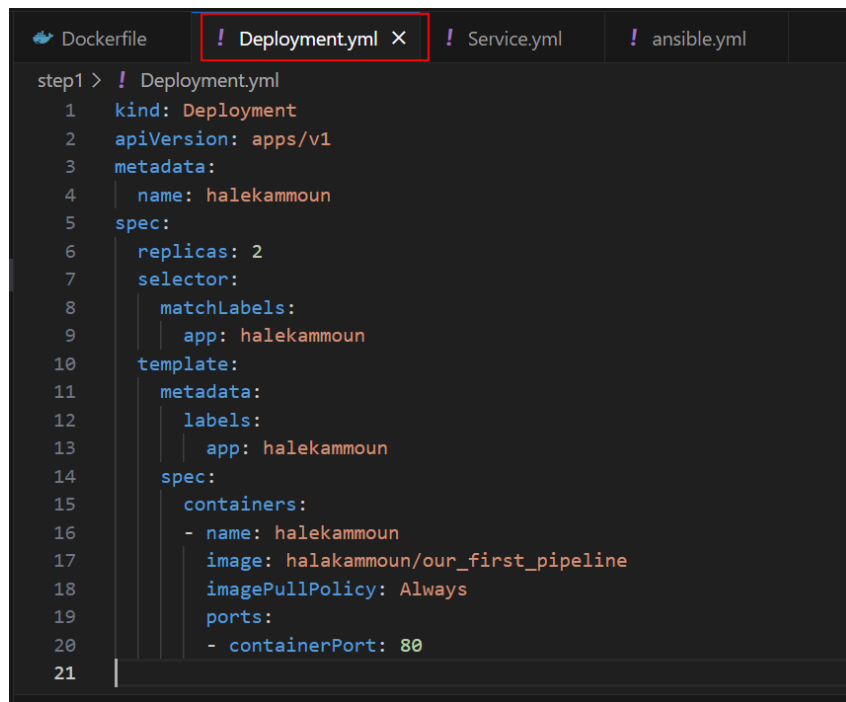
Cela garantit que les images taggées sont publiées avec succès sur Docker Hub et que l'image "latest" est toujours synchronisée avec la version la plus récente construite par Docker.

Le sixième stage « Copy files from ansible to k8s »:

Nous avons tous d'abord créé les fichiers Deployment.yml, service.yml et ansible.yml.

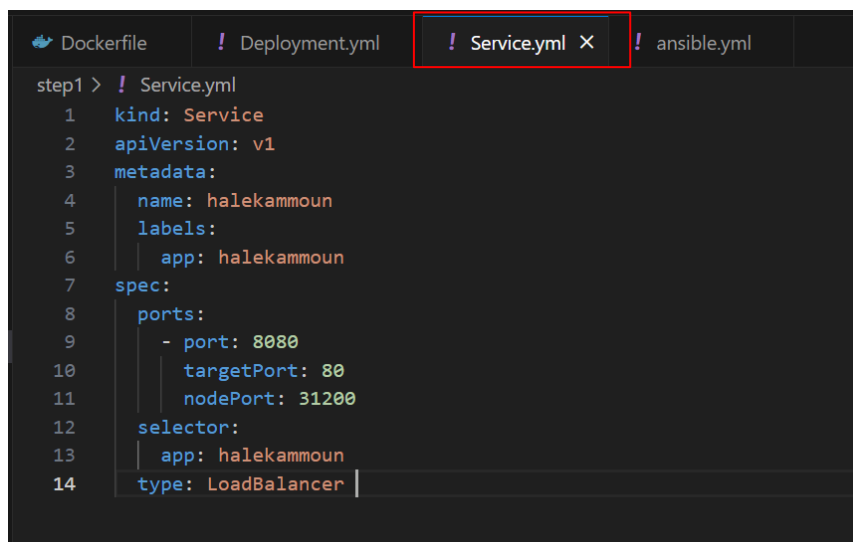
Ensuite, Nous avons ajouté les configurations nécessaires à ces fichiers comme montre les figures 22, 23 et 24 suivantes.

Puis, nous avons effectué un "git push" pour sauvegarder les modifications.



```
step1 > ! Deployment.yml X ! Service.yml ! ansible.yml
1 kind: Deployment
2 apiVersion: apps/v1
3 metadata:
4   name: halekammoun
5 spec:
6   replicas: 2
7   selector:
8     matchLabels:
9       app: halekammoun
10  template:
11    metadata:
12      labels:
13        app: halekammoun
14    spec:
15      containers:
16      - name: halekammoun
17        image: halekammoun/our_first_pipeline
18        imagePullPolicy: Always
19        ports:
20        - containerPort: 80
21
```

Figure 22: Contenu de playbook Deployment.yml



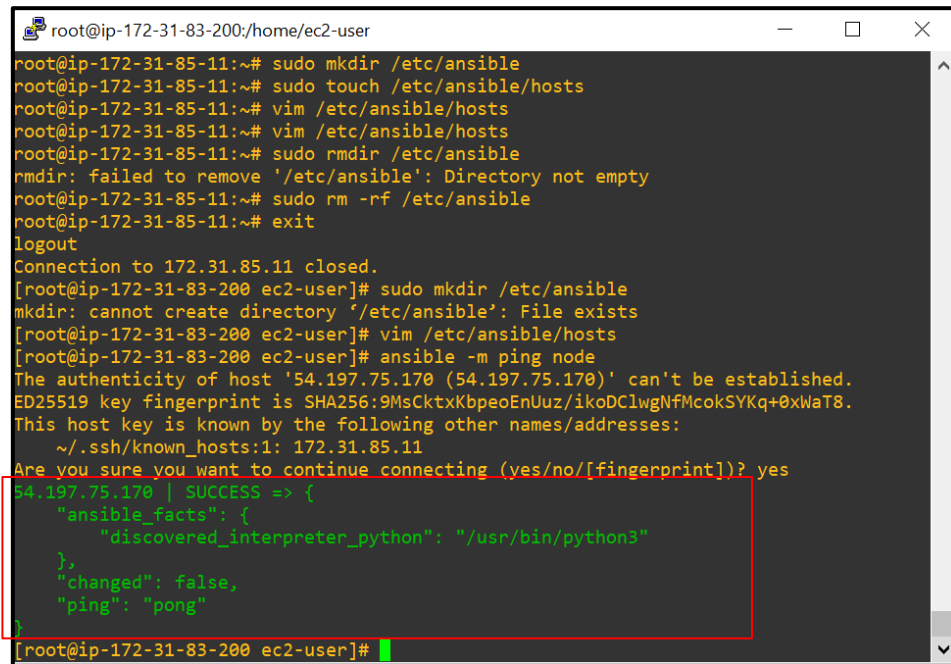
```
step1 > ! Deployment.yml X ! Service.yml X ! ansible.yml
1 kind: Service
2 apiVersion: v1
3 metadata:
4   name: halekammoun
5   labels:
6     app: halekammoun
7 spec:
8   ports:
9   - port: 8080
10     targetPort: 80
11     nodePort: 31200
12   selector:
13     app: halekammoun
14   type: LoadBalancer
```

Figure 23: Contenu de playbook service.yml

Ensuite, nous avons ajouté l'adresse du serveur Kubernetes dans le fichier hosts d'Ansible :

Nous avons modifié le fichier hosts d'Ansible pour inclure l'adresse IP publique du serveur Kubernetes.

Pour vérifier la connectivité avec le serveur Kubernetes, nous avons utilisé la commande "ansible -m ping node" comme montre la figure 26 suivante.

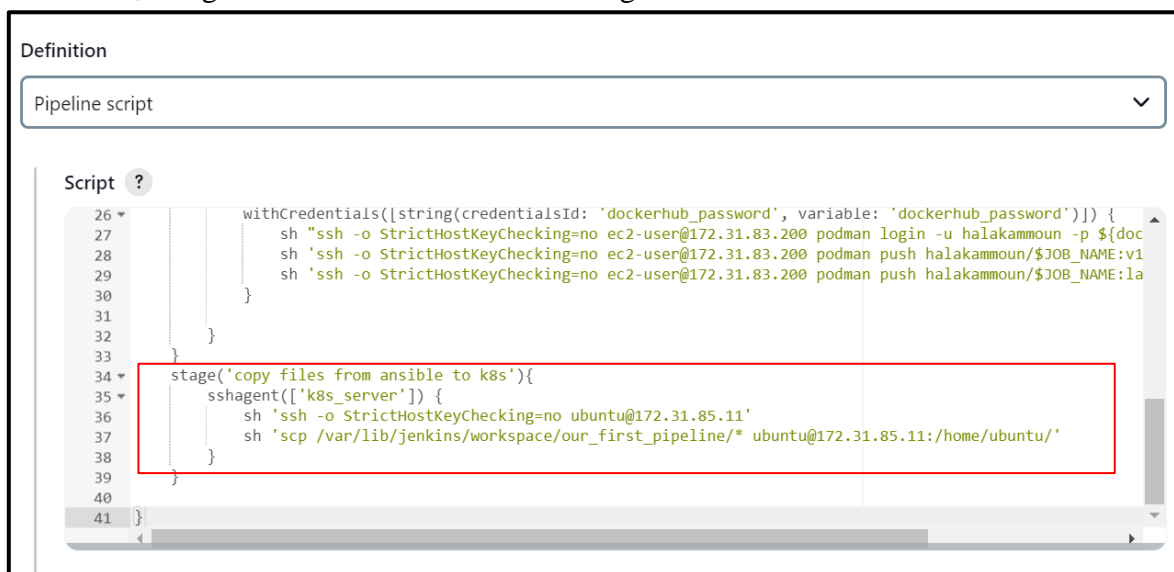


```
root@ip-172-31-83-200:/home/ec2-user
root@ip-172-31-85-11:~# sudo mkdir /etc/ansible
root@ip-172-31-85-11:~# sudo touch /etc/ansible/hosts
root@ip-172-31-85-11:~# vim /etc/ansible/hosts
root@ip-172-31-85-11:~# vim /etc/ansible/hosts
root@ip-172-31-85-11:~# sudo rmdir /etc/ansible
rmdir: failed to remove '/etc/ansible': Directory not empty
root@ip-172-31-85-11:~# sudo rm -rf /etc/ansible
root@ip-172-31-85-11:~# exit
logout
Connection to 172.31.85.11 closed.
[root@ip-172-31-83-200 ec2-user]# sudo mkdir /etc/ansible
mkdir: cannot create directory '/etc/ansible': File exists
[root@ip-172-31-83-200 ec2-user]# vim /etc/ansible/hosts
[root@ip-172-31-83-200 ec2-user]# ansible -m ping node
The authenticity of host '54.197.75.170 (54.197.75.170)' can't be established.
ED25519 key fingerprint is SHA256:9MsCktxKbpeoEnUuz/ikoDC1wgNfMcokSYKq+0xWaT8.
This host key is known by the following other names/addresses:
  ~/.ssh/known_hosts:1: 172.31.85.11
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
54.197.75.170 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
[root@ip-172-31-83-200 ec2-user]#
```

Figure 26: vérification de connectivité vers kubernetes.

Nous avons ajouté la clé privée du serveur Kubernetes aux identifiants de Jenkins.

Ensuite, nous avons configuré le stage dans Jenkins pour transférer les fichiers vers Kubernetes, la figure 27 montre le contenu de stage.



```
Definition
Pipeline script
Script ?
26 withCredentials([string(credentialsId: 'dockerhub_password', variable: 'dockerhub_password')]) {
27     sh "ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 podman login -u halakammoun -p ${doc
28     sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 podman push halakammoun/$JOB_NAME:v1
29     sh 'ssh -o StrictHostKeyChecking=no ec2-user@172.31.83.200 podman push halakammoun/$JOB_NAME:la
30 }
31 }
32 }
33 }
34 stage('copy files from ansible to k8s'){
35     sshagent(['k8s_server']) {
36         sh 'ssh -o StrictHostKeyChecking=no ubuntu@172.31.85.11'
37         sh 'scp /var/lib/jenkins/workspace/our_first_pipeline/* ubuntu@172.31.85.11:/home/ubuntu/'
38     }
39 }
40 }
41 }
```

Figure 27: configuration de sixième stage.

Pour réaliser le transfert des fichiers depuis Jenkins vers Kubernetes, nous avons utilisé la commande scp dans ce dernier stage.

Nous avons également généré une nouvelle paire de clés SSH sur Jenkins et l'avons envoyée au serveur Kubernetes la figure suivante démontre la réalisation de stage avec succès.

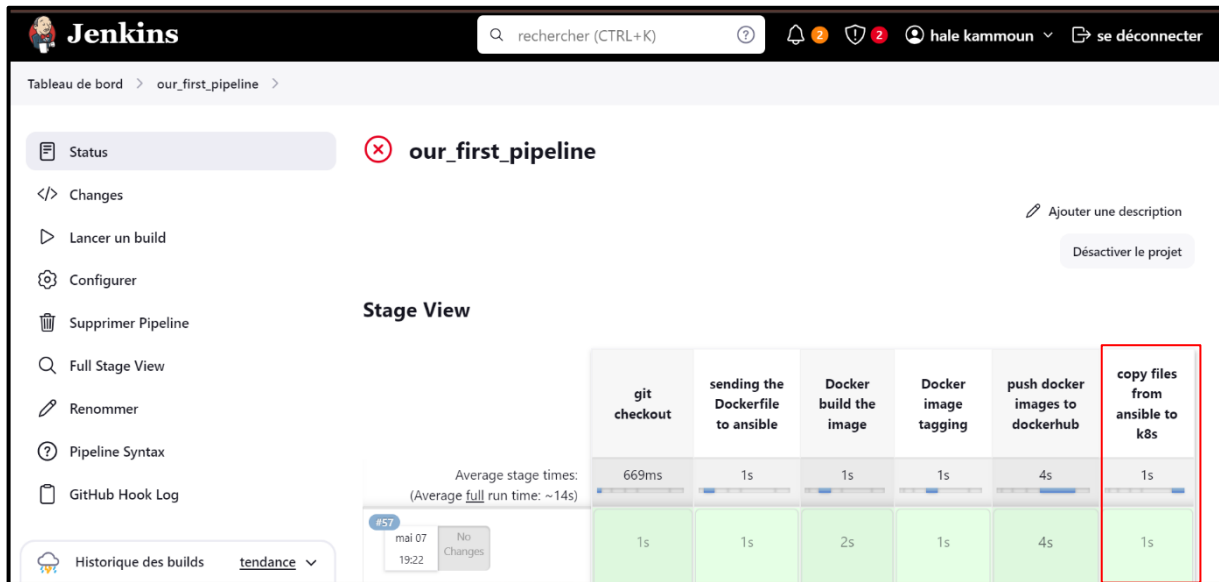


Figure 28: Exécution de sixième stage.

Septième stage « k8s deployment using ansible »:

Nous avons déjà configuré les tâches nécessaires dans le playbook Ansible pour le déploiement sur Kubernetes.

Enfin, nous avons exécuté le playbook ansible.yml pour déployer les applications sur le cluster Kubernetes comme montre les figure 29 et 30 suivantes.



Figure 29: configuration de septième stage.

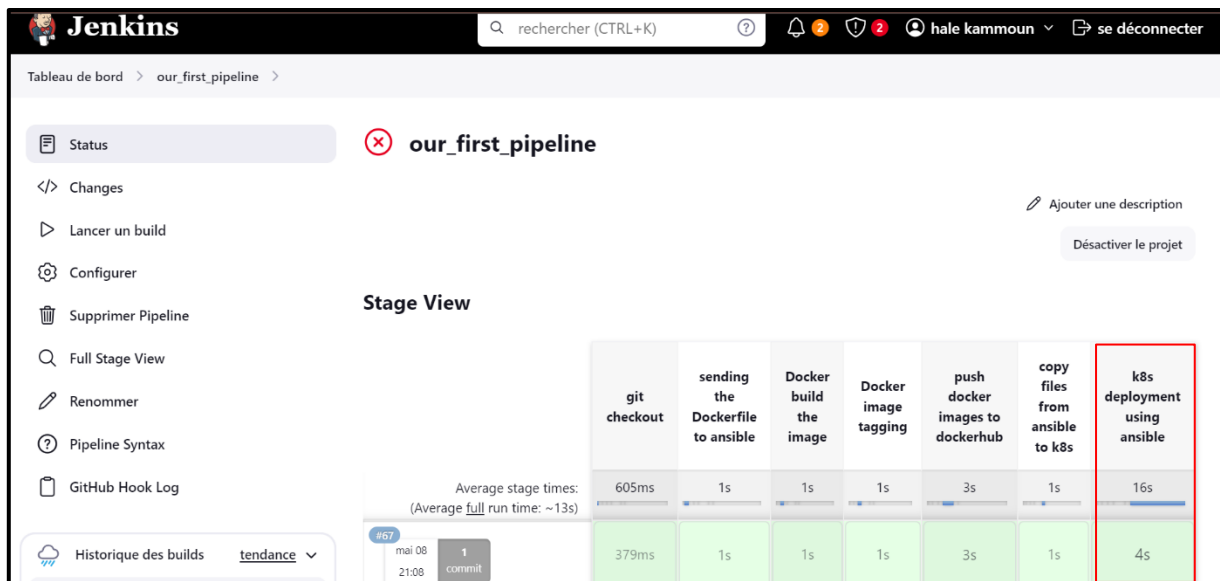


Figure 30: Exécution de septième stage.

Conclusion

En résumé, ce chapitre nous a plongés dans la mise en œuvre pratique de notre projet. Nous avons détaillé chaque étape, depuis la configuration des serveurs EC2 jusqu'à l'orchestration des déploiements sur Kubernetes. En utilisant une combinaison intelligente d'outils comme Docker, GitHub, Jenkins, Ansible et Kubernetes, nous avons automatisé le processus de développement et de déploiement, ce qui nous a permis d'accroître l'efficacité et la fiabilité de notre workflow. En fin de compte, notre objectif était de créer une solution DevOps robuste et performante, et ce chapitre représente une avancée majeure vers la réalisation de cet objectif.

Conclusion générale

Dans l'ensemble, ce projet représente une exploration approfondie et une mise en pratique des principes fondamentaux de DevOps pour améliorer le cycle de vie du développement logiciel. En combinant une gamme d'outils modernes et en adoptant une approche méthodique, nous avons réussi à transformer une vision théorique en une réalité opérationnelle.

À travers les différentes phases du projet, de la conception initiale à la réalisation concrète, nous avons démontré l'importance de l'automatisation, de la collaboration et de la mise en œuvre efficace des pratiques DevOps. En intégrant des outils tels que Docker, GitHub, Jenkins, Ansible et Kubernetes, nous avons créé un écosystème cohérent et interconnecté, permettant un déploiement continu et fiable des applications.

Les bénéfices de cette approche sont multiples : une réduction significative des délais de déploiement, une amélioration de la qualité du logiciel grâce à des tests automatisés et une meilleure gestion des environnements de développement et de production. De plus, cette démarche favorise une collaboration étroite entre les équipes de développement et d'exploitation, renforçant ainsi la culture DevOps au sein de notre organisation.

En conclusion, ce projet illustre le potentiel transformatif de DevOps dans le domaine du développement logiciel. En adoptant une approche centrée sur l'automatisation, la collaboration et l'amélioration continue, nous avons posé les bases d'une infrastructure robuste et adaptable, capable de répondre aux défis futurs de manière agile et efficace.

Webographie

[1] Présentation de TEK-UP University.

<https://tek-up.de/study-at-tek-up-university/>

[2] Documentation Ansible

<https://docs.ansible.com/>

[3] Site officiel de Docker

<https://www.docker.com/>

[4] Tutoriel Docker pour débutants

<https://www.freecodecamp.org/news/what-is-docker-learn-how-to-use-containers-with-examples/>

[5] Site officiel de GitHub

<https://github.com/>

[6] Cours Git et Github pour débutants

<https://www.freecodecamp.org/news/git-and-github-crash-course/>

[7] Site officiel de Jenkins

<https://www.jenkins.io/>

[8] Tutoriel Jenkins pour débutants

<https://www.freecodecamp.org/news/tag/jenkins/>

[10] Introduction à Ansible

<https://www.freecodecamp.org/news/tag/ansible/>

[11] Site officiel de Kubernetes

<https://kubernetes.io/>

[12] Guide Kubernetes pour débutants

<https://www.freecodecamp.org/news/the-kubernetes-handbook/-differents>