



FACULTÉ  
DES SCIENCES  
DE TUNIS

République Tunisienne  
Ministère de l'Enseignement Supérieur  
et de la Recherche Scientifique  
Université de Tunis El Manar  
Faculté de Science de Tunis



## MÉMOIRE DE MASTÈRE

Présenté en vue de l'obtention du  
Diplôme Nationale de Mastère Professionnel en Systèmes de Télécommunications & Réseaux  
Spécialité : Mise en Réseau de Systèmes Informatiques et Télécommunications

Par

**Mohamed Akram LARAFA**

---

### Mise en place d'un pipeline d'intégration et de déploiement continus à base de Kubernetes

---

Réalisé au sein de CIAM Solution



Soutenu publiquement le 31/01/2024 devant le jury composé de :

Président : Mme Imen CHERIF  
Examinateur: M Anouar BEN MESSAOUD  
Encadrant académique : M Faker LANDOLSI  
Encadrant professionnel : M Mehdi Ben LAKHAL

Année Universitaire 2022 – 2023



J'autorise l'étudiant à faire le dépôt final de son rapport de stage en vue d'une soutenance.

Encadrant académique, Monsieur **Faker LANDOLSI**

Signature

J'autorise l'étudiant à faire le dépôt final de son rapport de stage en vue d'une soutenance.

Président du jury, Madame **Imen CHERIF**

Signature

## Dédicaces

À mes chers parents, mes idoles, aucun dévouement ne suffit pour exprimer mon amour et mon respect pour vous.

Merci à mes frères Ahmed et Maher et à ma sœur Mariam qui ne cessent de m'encourager et de me soutenir dans tous les projets sur lesquels je travaille. En témoignage de ma fraternité, de mon attachement éternel et de ma gratitude.

A toutes les personnes que j'aime.

À tous ceux qui m'aiment et me soutiennent.

**Mohamed Akram LARAFA**

## Remerciement

Au terme de ce stage, je tiens tout d'abord à remercier CIAM Solution de m'avoir proposé ce stage.

Ensuite, j'exprime ma gratitude envers Monsieur Mehdi Ben Lakhel, qui m'a encadré pendant mon projet de fin d'étude. Je le remercie pour sa disponibilité et l'intérêt qu'il a porté pour ce travail.

J'adresse mes chaleureux remerciements à Monsieur Faker LANDOLSI pour ses précieux conseils, son aide et l'effort qu'elle n'a pas ménagé pour m'aider à rédiger ce rapport.

Mohamed Akram LARAFA

## Table des matières

Remerciement.....	ii
Table des figures .....	vii
Liste des Tableaux.....	iii
Liste des abréviations .....	iv
Introduction Générale.....	1
<b>1. CADRE GÉNÉRAL.....</b>	<b>3</b>
1.1. Introduction .....	3
1.2. Organisme d'accueil .....	3
1.2.1. Activités et produits de CIAM Solution .....	3
1.2.2. Département d'accueil .....	3
1.3. Étude de l'existant et constatations .....	4
1.4. Objectifs et contributions.....	5
1.5. Méthodologie de travail.....	7
1.5.1. Méthode de gestion de projet.....	7
1.5.2. Méthodologie .....	9
1.6. Conclusion .....	11
<b>2. ÉTUDE PRÉALABLE .....</b>	<b>12</b>
2.1. Introduction .....	12
2.2. Cloud Computing .....	12
2.2.1. Définition et généralités .....	12
2.2.2. Les modèles de déploiement de cloud computing .....	12
2.2.3. Les modèles de cloud computing.....	13
2.2.4. Avantages du cloud computing.....	15
2.2.5. Les principaux fournisseurs .....	16
2.3. Intégration continue .....	17
2.4. Livraison continue .....	18
2.5. Déploiement continu.....	18

## Table des matières

---

2.6.	Éléments clés .....	18
2.6.1.	Kubernetes .....	18
2.6.2.	Pod .....	19
2.6.3.	Docker.....	19
2.6.4.	Container.....	19
2.6.5.	Cluster.....	19
2.6.6.	Kustomize .....	20
2.6.7.	Runner.....	20
2.6.8.	HELM .....	20
2.6.9.	Portainer.....	20
2.6.10.	Ingress.....	21
2.6.11.	Cert-Manager.....	21
2.7.	Étude comparative .....	21
2.7.1.	Gestion de versions : Gitlab vs SVN .....	21
2.7.2.	Outils d'orchestration : GitLab CI vs Jenkins.....	22
2.7.3.	Virtualisation : Docker vs Machine virtuelle .....	24
2.7.4.	Plateforme de déploiement : Kubernetes vs Docker Swarm .....	24
2.7.5.	Supervision : Prometheus & Grafana vs Zabbix.....	26
2.7.6.	SonarQube vs CodeScan.....	27
2.7.7.	Outil de configuration et d'approvisionnement : Terraform vs Ansible.....	27
2.8.	Synthèse et choix .....	28
2.9.	Conclusion .....	28
<b>3.</b>	<b>ANALYSE ET SPÉCIFICATIONS .....</b>	<b>29</b>
3.1.	Introduction .....	29
3.2.	Détermination des acteurs .....	29
3.3.	Analyse des besoins.....	30
3.3.1.	Détermination des exigences fonctionnelles.....	30

3.3.2. Détermination des exigences techniques .....	30
3.3.3. Les exigences non fonctionnelles .....	31
3.4. Représentation des exigences .....	31
3.4.1. Intégration continue .....	33
3.4.2. Déploiement continu .....	36
3.4.3. Configuration .....	39
3.4.4. Supervision .....	41
3.4.5. Pré-configuration .....	43
<b>4. Architecture et conception détaillée .....</b>	<b>51</b>
4.1. Introduction .....	51
4.2. Architecture générale de la solution .....	51
4.3. Aspect dynamique de la solution.....	53
4.3.1. Vue dynamique du volet "Intégration continue".....	53
4.3.2. Vue dynamique du volet "Déploiement continu" .....	54
4.3.3. Vue dynamique du volet "Configuration" .....	57
4.3.4. Vue dynamique du volet "Pré-configuration" .....	59
4.4. Conclusion .....	66
<b>5. RÉALISATION.....</b>	<b>67</b>
5.1. Introduction .....	67
5.2. Environnement de travail.....	67
5.2.1. Environnement Microsoft Azure : .....	67
5.2.2. Plateforme GitLab SaaS :.....	67
5.2.3. Environnement local : .....	68
5.3. Technologies utilisées.....	68
5.3.1. Configuration déclarative: YAML.....	68
5.3.2. Plateforme de déploiement: Kubernetes .....	68
5.3.3. Technologie de configuration et de provisionnement : Terraform .....	68
5.3.4. Gitlab.....	69

## Table des matières

---

5.3.5. Serveur d’application .....	73
5.3.6. Outil de test : Sonarqube.....	74
5.3.7. Outil de build d’image docker : Kaniko.....	74
5.4. Principaux développements .....	74
5.4.1. Intégration et déploiement continus .....	75
5.4.2. Configuration .....	78
5.4.3. Supervision .....	89
5.4.4. Pré-configuration .....	92
5.5. Conclusion .....	100
Conclusion Générale .....	101
Netographie .....	103

## Table des figures

Figure 1 : Clients de CIAM Solution .....	4
Figure 2 : Illustration de la chaîne CI/CD chez CIAM Solution.....	4
Figure 3 : Processus DevOps .....	5
Figure 4 : Schéma illustrant la solution proposée. ....	6
Figure 5 : Architecture générale - les micros services utilisées .....	7
Figure 6 : Une itération selon la méthode Scrum.....	8
Figure 7 : Le processus « 2TUP » .....	10
Figure 8 : Les différents modèles de services cloud .....	14
Figure 9 : Fournisseurs Cloud: GCP vs Azure vs AWS .....	17
Figure 10 : Logo Kubernetes.....	19
Figure 11 : Logo Docker .....	19
Figure 12 : Logo Gitlab Runner .....	20
Figure 13 : Logo HELM.....	20
Figure 14 : Container vs VM.....	24
Figure 15 : Diagramme général du système .....	32
Figure 16 : Illustration du diagramme de partie "Intégration continue" .....	34
Figure 17 : Diagramme du cas d'utilisation "Déploiement continu" .....	37
Figure 18 : Diagramme de cas d'utilisation "Configuration".....	39
Figure 19 : Méthodes de déploiement utilisées .....	40
Figure 20 : Diagramme de cas d'utilisation "Supervision".....	42
Figure 21 : Diagramme de cas d'utilisation "Création et configuration d'un cluster AKS" ....	44
Figure 22 : Architecture de déploiement et d'approvisionnement d'AKS avec Terraform .....	46
Figure 23 : Diagramme de cas d'utilisation "Création et configuration d'un compte Gitlab".	46
Figure 24 : Diagramme d'architecture générale.....	51
Figure 25 : Diagramme de déploiement et d'approvisionnement pour l'architecture à base de Kubernetes.....	52
Figure 26 : Processus d'Intégration continue .....	54
Figure 27 : Processus de déploiement d'une application .....	56
Figure 28 : Processus de création et définition d'un projet .....	58
Figure 29 : Processus de mise en place et approvisionnement d'un Cluster AKS .....	60
Figure 30 : Configuration de la sécurité, de la supervision et de l'accès.....	61
Figure 31 : Architecture de la solution de supervision des ressources.....	62

## Table des figures

---

Figure 32 : Processus de configuration de supervision .....	63
Figure 33 : Processus de création et configuration d'un compte Gitlab .....	64
Figure 34 : Processus générale du pipeline CI/CD .....	65
Figure 35 : Architecture globale du Gitlab.....	69
Figure 36 : Processus d'une tâche de pipeline GitLab standard .....	71
Figure 37 : Processus d'intégration continue CI .....	72
Figure 38 : Processus de déploiement continu CD .....	73
Figure 39 : Processus d'intégration et livraison continue .....	75
Figure 40 : Processus de déploiement continu .....	76
Figure 41 : Résultat du scan et analyse du code source .....	76
Figure 42 : Résultat de l'exécution du job de Build .....	77
Figure 43 : Liste des images créées et hébergées dans le registre Gitlab.....	77
Figure 44 : Déploiement de l'application dans l'environnement de développement .....	78
Figure 45 : Dashboard du Portainer .....	79
Figure 46 : Déploiement et configuration du Ingress-Controller .....	80
Figure 47 : Déploiement d'émetteur Lets Encrypt .....	81
Figure 48 : Déploiement et configuration de Sonarqube .....	81
Figure 49 : Sonarqube intégré avec Gitlab .....	82
Figure 50 : Définition d'une Dockerfile .....	82
Figure 51 : Configuration d'accès pour un ServiceAccount.....	83
Figure 52 : Déploiement et configuration de PostgreSQL.....	84
Figure 53 : Déploiement et configuration de pgAdmin .....	85
Figure 54 : Dashboard de pgAdmin 4 .....	85
Figure 55 : Déploiement d'instance Odoo .....	86
Figure 56 : Interface des applications Odoo.....	86
Figure 57 : Liste des étapes du pipeline .....	87
Figure 58 : Définition de l'étape « Scanner » du pipeline .....	87
Figure 59 : Définition de l'étape « Build » du pipeline .....	88
Figure 60 : Définition de l'étape « Deploy » du pipeline .....	88
Figure 61 : Définition de l'étape « deploy-prod » du pipeline .....	89
Figure 62 : Dashboard usage CPU et RAM dans le cluster de production .....	90
Figure 63 : Utilisation du CPU par l'instance Odoo en production.....	90
Figure 64 : Utilisation de RAM par l'instance Odoo en production.....	91
Figure 65 : Utilisation du réseau par l'instance Odoo en production .....	91

## Table des figures

---

Figure 66 : Définition des clusters avec des modules Terraform .....	92
Figure 67 : Définition de la création de kubeconfig à l'aide de local_file .....	93
Figure 68 : Cluster AKS de l'environnement de développement .....	93
Figure 69 : Cluster AKS de l'environnement de production .....	94
Figure 70 : Liste des ressources créées pour chaque cluster AKS .....	94
Figure 71 : Identification du compte Gitlab .....	95
Figure 72 : Création d'un agent au niveau Gitlab .....	96
Figure 73 : Les instructions d'installation d'agent Gitlab sur l'AKS .....	96
Figure 74 : Installation de l'agent sur le cluster AKS .....	97
Figure 75 : Gitlab plateforme connecté à l'AKS .....	97
Figure 76 : Le Runner est assigné à notre projet Gitlab .....	98
Figure 77 : Connexion établie entre Sonarqube et Gitlab .....	99

## Liste des Tableaux

Tableau 1 - Backlog produit.....	9
Tableau 2 - Tableau de comparaison Gitlab vs SVN .....	22
Tableau 3 - Tableau de comparaison Gitlab CI vs Jenkins .....	23
Tableau 4 - Tableau de comparaison Kubernetes vs Docker Swarm.....	26
Tableau 5 - Tableau de comparaison Prometheus & Grafana vs Zabbix.....	26
Tableau 6 - Tableau de comparaison CodeScan vs SonarQube.....	27
Tableau 7 - Tableau de comparaison Terraform vs Ansible .....	28
Tableau 8 - les choix technologiques .....	28
Tableau 9 - Description textuelle du cas d'utilisation "Déposer code".....	34
Tableau 10 - Description textuelle du cas d'utilisation "Récupération du code".....	35
Tableau 11 - Description textuelle du cas d'utilisation "Déclencher un scan du code".....	35
Tableau 12 - Description textuelle du cas d'utilisation "Déclencher un build" .....	36
Tableau 13 - Description textuelle du cas d'utilisation "Déployer application" .....	37
Tableau 14 - Description textuelle du cas d'utilisation "Récupérer image" .....	38
Tableau 15 - Description textuelle du cas d'utilisation "Récupérer l'URL de production"....	38
Tableau 16 - Description textuelle du cas d'utilisation "Récupérer l'URL du test" .....	39
Tableau 17 - Description textuelle du cas d'utilisation "Préparation des configurations de déploiement".....	41
Tableau 18 - Description textuelle du cas d'utilisation "Surveiller le cluster et les projets" ...	42
Tableau 19 - Description textuelle du cas d'utilisation "Surveiller l'état des Pods" .....	43
Tableau 20 - Description textuelle du cas d'utilisation "Mise en place un cluster AKS".....	45
Tableau 21 - Description textuelle du cas d'utilisation "Création et configuration d'un compte gitlab.com" .....	47
Tableau 22 - Description textuelle du cas d'utilisation "Ajouter un cluster Kubernetes à gitlab.com" .....	47
Tableau 23 - Description textuelle du cas d'utilisation "Ajouter un Runner à gitlab.com".....	48
Tableau 24 - Description textuelle du cas d'utilisation "Intégrer SonarQube à gitlab.com" ...	49
Tableau 25 - Description textuelle du cas d'utilisation "Configurer l'analyse pour un projet".....	50
Tableau 26 - Description textuelle du cas d'utilisation "Fournir l'accès aux chefs des projets" .....	50
Tableau 27 - Ressources réservées à un nœud du cluster AKS.....	67
Tableau 28 - Les portails utilisé et configuré .....	89

## Liste des abréviations

- azurerm = Azure Resource Manager
- CD = Continuous Delivry
- CI = Continuous Integration
- JSON = JavaScript Object Notation
- SSL = Secure Sockets Layer
- SRE = Software Reliability Engineer
- TLS = Transport Layer Security
- YAML = Yet Another Markup Language
- UML = Unified Modeling Language
- SCM = Source Code Management
- CLOUD Act = Clarifying Lawful Overseas Use of Data Act
- RGPD = Règlement Général sur la Protection des Données

## Introduction Générale

D'une manière générale, la mise en œuvre d'une application en production est l'étape finale d'un processus bien développé qui implique souvent différentes équipes, à savoir les équipes opérationnelles et fonctionnelles qui sont responsables des tests et du développement. Par conséquent, ces équipes sont souvent considérées comme trois phases distinctes du processus.

L'implication d'un grand nombre d'équipes peut être une source de conflit, car chaque une peut avoir des objectifs différents. Lorsque les développeurs veulent innover et développer les applications, la principale préoccupation de l'équipe de production est de maintenir la stabilité du système informatique. De plus, chacun suit ses propres méthodes et outils de travail, avec peu de communication. De ce fait, des conflits de communication entre les membres de l'équipe peuvent survenir, et ces conflits peuvent entraîner des inquiétudes sur les délais, davantage de demandes à traiter, et par conséquent une augmentation des coûts pour l'entreprise qui n'étaient pas initialement prévus. Cela affecte la satisfaction des clients, qui est au cœur des préoccupations de l'entreprise.

Il est donc devenu évident que nous avions besoin d'une nouvelle approche pour automatiser et d'intégrer les processus entre les équipes de développement et informatiques afin d'éviter tous les problèmes ci-dessus.

Par conséquent, Le mouvement DevOps a apparu vers 2007, Le terme DevOps, qui combine les mots « développement » et « opérations », reflète le processus d'intégration de ces disciplines en un processus continu. Les équipes DevOps comprennent des développeurs et des experts en opérations informatiques qui collaborent tout au long du cycle de vie du produit pour rendre le déploiement de logiciels plus rapide et de meilleure qualité. C'est une nouvelle façon de travailler, un changement culturel qui a des conséquences importantes pour les équipes et les organisations qui la pratiquent.

L'objectif de ce projet de fin d'études au CIAM est de mettre en place une solution capable de rassembler toutes les équipes autour d'une même approche, et ce à travers une plateforme DevOps, comprenant des pipelines d'intégration et de déploiement continus, avec un tableau de bord de surveillance des composants du système.

Notre rapport est divisé en cinq chapitres. Nous commençons par l'introduction du premier chapitre, intitulé "Cadre générale", dans lequel nous présentons l'organisation d'accueil,

introduisons la solution existante la méthode utilisée, nous poursuivons avec les objectifs et la solution proposée pour le projet, et enfin avec la méthodologie de développement adoptée.

Le chapitre 2 "Étude préalable" nous permet de définir les environnements et les techniques utilisés dans notre projet, puis de clarifier les éléments clés liés aux outils et aux technologies que nous étudierons, comparerons et résumerons dans la dernière partie de ce chapitre.

Dans le chapitre 3, "Analyse et spécification", nous avons déterminé les acteurs du système, en précisant les différentes exigences techniques, fonctionnelles et non fonctionnelles. Nous procédons ensuite à la représentation de ces besoins.

Le chapitre 4, "Architecture et conception détaillée", qui présente l'architecture globale de notre solution et sa vue dynamique.

Le chapitre "Réalisation" nous permet de montrer les plateformes et les logiciels sur lesquels nous travaillons, de montrer les différentes technologies que nous utilisons pour mettre en œuvre notre travail, et enfin, nous montrons les résultats de l'exécution des jobs et des déploiements sous forme de captures d'écran.

Nous terminons notre rapport par une conclusion générale qui donne quelques réflexions pour améliorer la solution.

# Chapitre 1

## CADRE GÉNÉRAL

### 1.1. Introduction

Dans ce premier chapitre. Nous commençons par une introduction de l'organisme d'accueil, puis nous examinons la situation existante et faisons des observations. Avant de conclure avec la méthodologie de développement, nous continuerons à expliquer les objectifs du projet.

### 1.2. Organisme d'accueil

CIAM Solutions est une entreprise experte en nouvelles technologies et en informatique, fondée en 2021. Elle maîtrise les méthodologies Cloud-Computing et DevOps, l'intégration logicielle et l'architecture de solutions.

CIAM se compose d'une équipe de développement des logiciels et une équipe d'opération spécialisé dans l'intégration des applications, gestion des infrastructures cloud, désigne des architectures cloud ainsi que l'implémentation des méthodologies DevOps aux clients.

#### 1.2.1. Activités et produits de CIAM Solution

Les quatre principaux domaines d'activité de CIAM Solution sont les suivants : Cloud-computing : mise en place et gestion d'infrastructures cloud ; DevOps : conception d'architectures cloud et mise en place de méthodologies DevOps à l'aide de solutions open source ; mise en place et intégration de solutions et d'applications ; et développement d'applications (Modules) basées sur le framework Odoo.

#### 1.2.2. Département d'accueil

Nous faisions partie du département d'ingénierie pendant le stage. Les besoins des clients, qu'il s'agisse de services de support ou d'intégration, sont au centre des efforts de cette équipe.



Figure 1 : Clients de CIAM Solution

### 1.3. Étude de l'existant et constatations

Dans un premier temps, nous présenterons la procédure et la méthode que l'équipe de développement de CIAM Solution a suivie initialement pour déployer les applications (modules Odoo), puis nous présenterons le problème.

Avant le processus de déploiement, CIAM utilise une approche simple de gestion du cycle de vie de ses applications, basée sur l'outil de gestion de versions « GIT », un serveur « Préprod » qui présente un environnement similaire au serveur de « Production », il est utilisé par l'équipe développeurs afin de faire les tests et validation des modules réalisé, enfin un serveur « Prod » qui va recevoir la nouvelle version du code source.

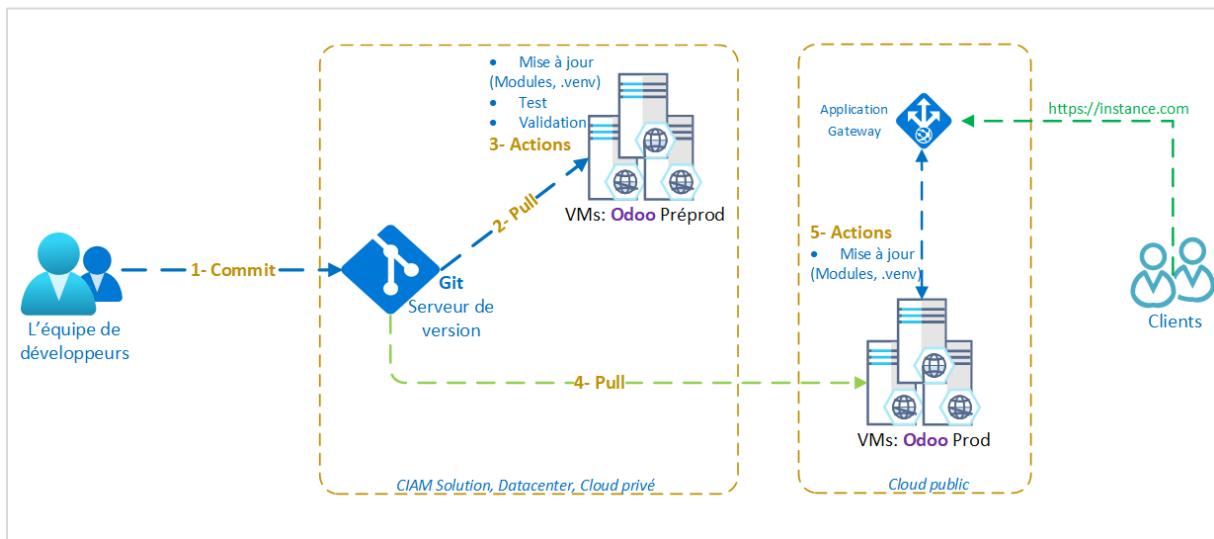


Figure 2 : Illustration de la chaîne CI/CD chez CIAM Solution

Avec l'architecture située dans la figure 2, la société CIAM Solution va trouver des difficultés afin de répondre aux cahiers des charges des projets, soit au niveau de la partie infrastructure soit pour l'approche de la gestion du projet et méthodologie de travail, nous citons ci-dessous les contraintes et les limites:

### Les limites de l'infrastructure :

- L'absence d'élasticité dynamique des ressources
- L'absence de redondance des données, en cas d'un désastre
- L'absence de l'auto-scaling et du contrôle des instances applicative

### Limite de méthode de travail :

- Le cycle de vie du produit est inachevé jusqu'à ce qu'il soit déployé auprès des clients.
- Absence d'une solution d'intégration continue.
- Les tests nécessaires, tels que les tests unitaires et les mesures de la qualité du code, ne sont pas effectués avant la livraison du produit.

### 1.4. Objectifs et contributions

L'objectif de notre travail est de créer d'un pipeline CI/CD (Intégration Continue & Déploiement Continu) dans l'environnement Azure cloud, ainsi que la construction et la configuration d'une interface de surveillance intuitive capable de nous fournir l'état de notre système et de ses actifs en termes de performance et de disponibilité.



Figure 3 : Processus DevOps

L'architecture que nous proposons est une solution clé en main de chaîne DevOps pour le développement, la production et la maintenance de l'ensemble de l'exposition. Elle implique la création d'une culture de « Infrastructure as a Code » avec une collaboration continue et flexible entre les développements, les opérations, et les équipes de sécurité. Elle implique l'utilisation de la philosophie DevOps, elle se concentre sur la création de nouvelles solutions pour des processus de développement logiciel complexe dans un cadre agile et sécurisé.

Notre pipeline d'intégration continu s'appuie sur des composants libres et propriétaires, dont la robustesse, la qualité et la pérennité ont été soigneusement étudiés, et l'attention a été portée sur la possibilité de répartir ses outils sur plusieurs plateformes et technologies, afin de permettre une souplesse suffisamment grande.

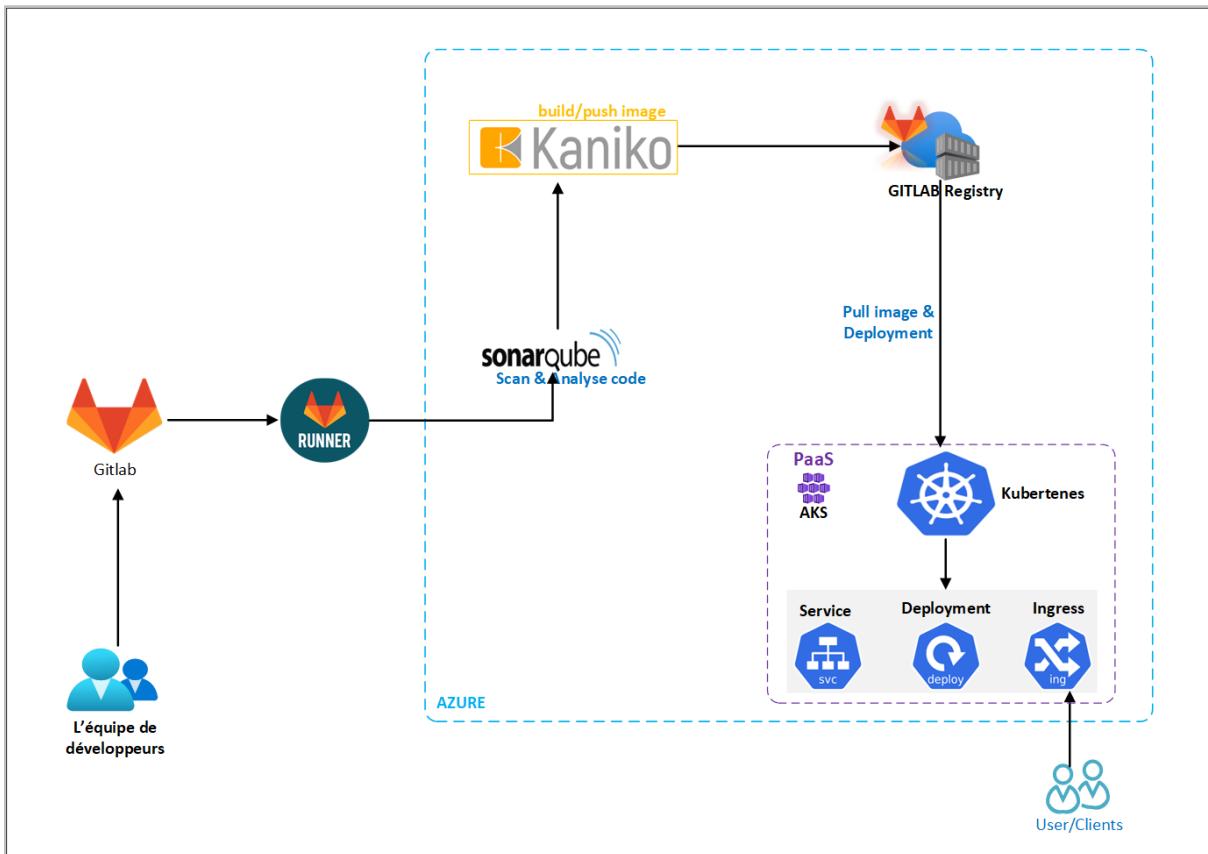


Figure 4 : Schéma illustrant la solution proposée.

La partie Code/Build/Test/Deploy sera accordée à la plateforme **Gitlab** intégrant sa propre solution d'intégration et déploiement continu nommée **Gitlab-Ci**. Il générera donc l'ensemble du cycle de vie DevOps qui permet aux équipes de mieux travailler ensemble et d'apporter plus de valeur aux clients, plus rapidement.

Pour s'assurer que l'équipe de développeurs Odoo se conforme à un ensemble uniforme de normes de codage, nous utilisons **SonarQube** pour l'analyse statique du code. Pour la construction de l'image Docker, nous avons choisi l'outil Kaniko.

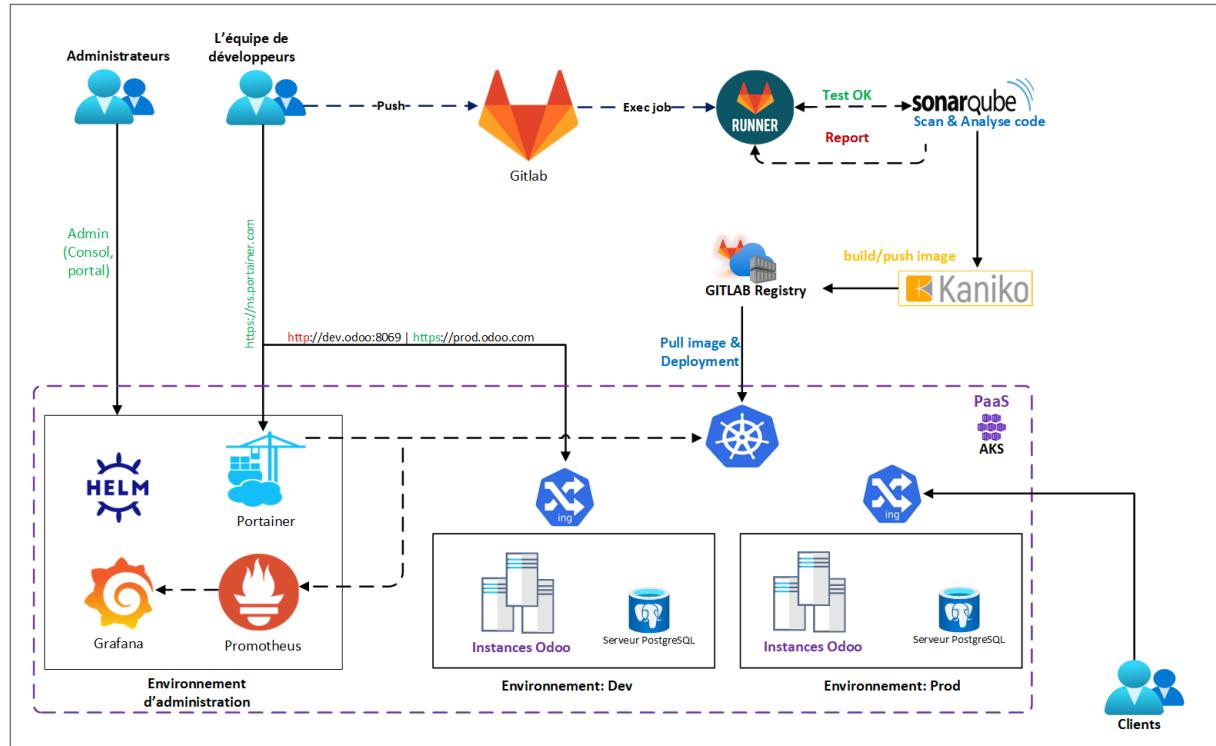


Figure 5 : Architecture générale - les microservices utilisés

Pour le déploiement des applications et leurs configurations nous avons choisi **Azure Kubernetes Service (AKS)** qui offre le moyen le plus rapide de commencer à développer et à déployer des applications natives cloud dans Azure.

Afin de garantir une bonne qualité de ces services, la partie monitor sera gérée par la solution **Grafana** et **Prometheus**. Cette étape supervisera les différents serveurs et services inclus dans notre architecture.

## 1.5. Méthodologie de travail

Les méthodes de travail utilisées pour gérer et réaliser le projet sont expliquées dans cette section.

### 1.5.1. Méthode de gestion de projet

Pour la gestion de différentes phases de notre projet, nous suivons une démarche inspirée de Scrum. Scrum fait partie des méthodes agiles de gestion de projet. L'objectif est donc d'augmenter la productivité des équipes agiles même à distance, tout en permettant l'optimisation des produits grâce à des retours réguliers avec les utilisateurs finaux. [1]

La figure 6 illustre le fonctionnement de la méthode Scrum. Une réunion d'équipe est organisée chaque jour pour évaluer ce qui a été fait et ce qui reste à faire. La durée du sprint est d'une semaine. Un sprint présente un morceau de produit.

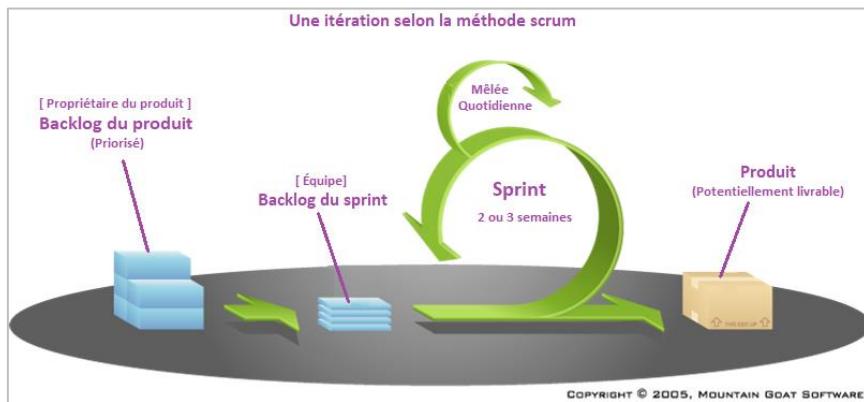


Figure 6 : Une itération selon la méthode Scrum

Le tableau 1 présente le backlog produit, qui est une liste de fonctionnalités requises pour implémenter la solution qui doivent être livrées dans un ordre spécifique.

Numéro	Tâche	Description	Priorité	Difficulté	Durée
1	Utilisation de l'approche (IaC) pour créer un cluster AKS (Azure Kubernetes Service)	L'administrateur peut développer un script Terraform pour mettre en place un cluster AKS (Azure Kubernetes Service), fournir l'accès aux développeurs (Chefs des projets) sur des spécifiques espaces des noms (« dev » par exemple)	2	Assez difficile	3 semaines
2	Création d'un compte sur gitlab.com, configuration et intégration avec le cluster Kubernetes	En tant que administrateur je dois préparer l'environnement de mon projet (création et assignation des rôles, ..), attacher un Runner à notre compte Gitlab, connecter notre cluster Kubernetes à Gitlab, intégrer Sonarqube à Gitlab et fournir l'accès aux développeurs (Chefs des projets).	1	Assez difficile	4 semaines
3	Mettre en place une chaîne d'intégration continue.	En tant que administrateur, je peux mettre en place une chaîne d'intégration continue et collaborer avec	3	Moyen	3 semaines

		l'équipe développement sur des spécifications technique comme le scan des extra-addons.			
4	Soumettre et configurer le projet à une chaîne de déploiement continu	En tant qu'administrateur, je peux configurer et paramétriser une plateforme pour rendre accessible pour les jobs de déploiement.	4	Difficile	3 semaines
5	Superviser l'état du cluster et des pods.	En tant que administrateur, je peux configurer un Dashboard de monitoring.	5	Assez difficile	4 semaines

Tableau 1 - Backlog produit

### 1.5.2. Méthodologie

Nous avons estimé que la méthode « Y » ou « 2TUP » était mieux adaptée à notre projet. La méthode « 2TUP » est un processus unifié conçu pour apporter des réponses aux contraintes d'évolution fonctionnelles et techniques imposées dans le développement logiciel, qui repose sur le principe selon lequel tous les changements apportés aux logiciels peuvent être décomposés et traités en parallèle selon des aspects fonctionnelles et des aspects techniques. Le développement de logiciels repose sur ces deux branches, qui doivent être fusionnées. [2]

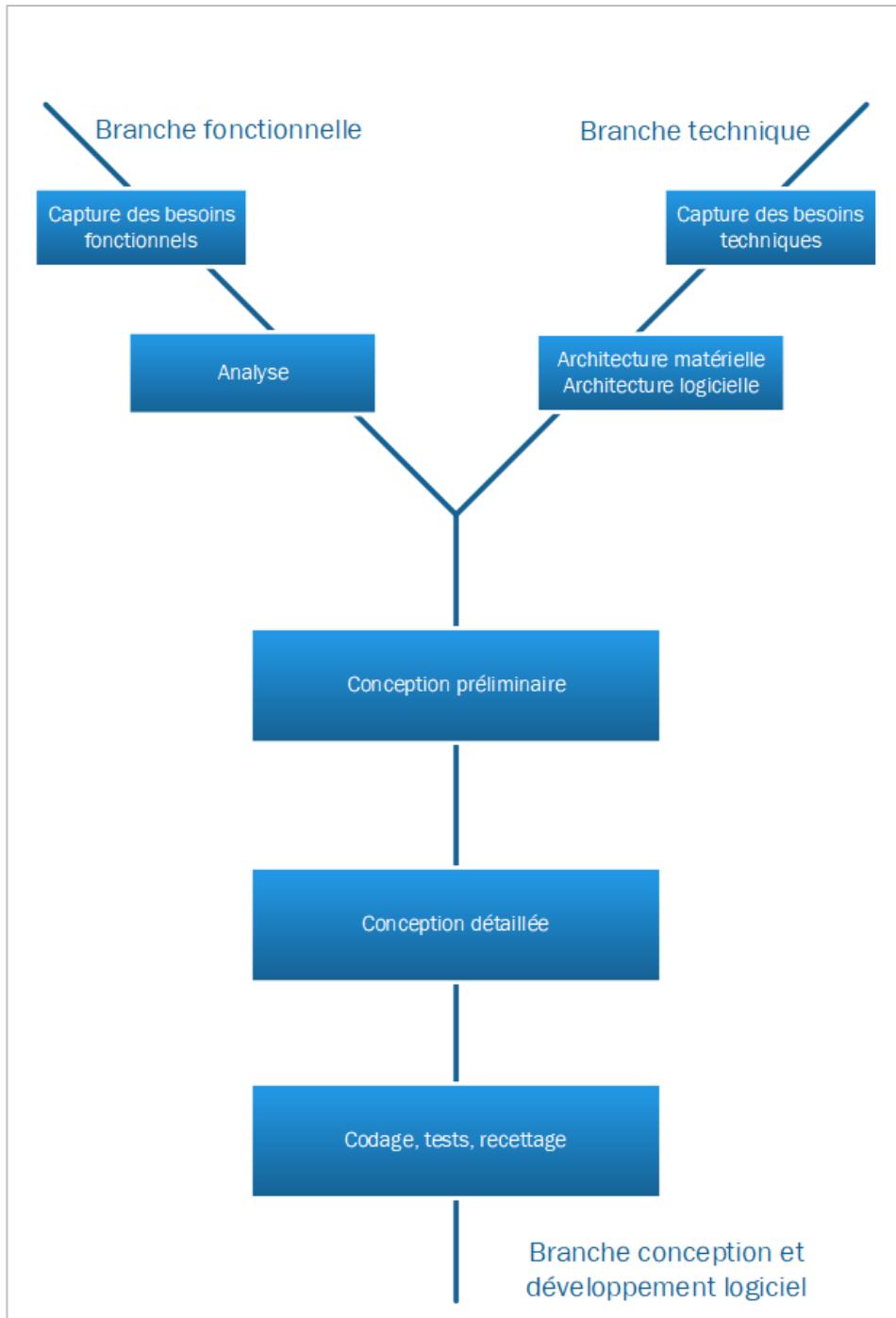


Figure 7 : Le processus « 2TUP »

- **Branche fonctionnelle** : Les exigences fonctionnelles et leur analyse sont saisies dans cette branche. À ce stade, la première étape consiste à déterminer les principales fonctions que le système doit remplir. Nous menons donc d'abord une étude des systèmes existants, en nous concentrant sur l'aspect fonctionnel qui nous permet de détecter leurs problèmes et de bien les comprendre. Pour éviter le risque de créer un système inapproprié, nous sommes également en mesure d'identifier les exigences fonctionnelles et non fonctionnelles des futurs systèmes.

- **Branche technique** : Cette branche est la partie qui définit les exigences techniques et spécifie l'architecture matérielle et logicielle. Dans cette section, nous identifions les prérequis matériels, tels que les caractéristiques techniques des serveurs et les logiciels utilisés pour mettre en place le pipeline.
- **Phase réalisation** : Cette phase est la fusion des deux branches précédentes, qui mène à la conception applicative adaptée aux besoins de notre solution. Cela implique la conception préliminaire, la conception détaillée, les phases de codage et de test et la phase d'acceptation. Dans notre cas, elle nous aide à préparer l'environnement de travail, mettre en place un pipeline CI/CD, ainsi qu'un système de surveillance de l'état des différents composants impliqués dans le pipeline.

Nous décrivons ces composants dans des cas d'utilisation en identifiant et en décrivant toutes les exigences de différentes natures.

## 1.6. Conclusion

Nous commençons ce chapitre par la présentation l'organisation d'accueil, puis nous avons examiné la situation existante et les résultats. Avant de conclure sur la méthodologie de développement, nous avons également défini l'objectif de notre projet.

# Chapitre 2

## ÉTUDE PRÉALABLE

### 2.1. Introduction

Dans ce qui suit, nous définissons d'abord le cloud computing, puis l'intégration et le déploiement continu, nous introduisons ensuite les éléments et concepts utilisés, nous passons à une étude comparative des plateformes et des outils, et enfin nous résumons nos choix technologiques.

### 2.2. Cloud Computing

#### 2.2.1. Définition et généralités

Le cloud computing consiste à fournir des services informatiques (comme des applications, des bases de données, des réseaux, du stockage et des serveurs) sur Internet. Cela signifie que les utilisateurs finaux peuvent accéder aux logiciels et aux applications, peu importe où ils se trouvent. Les systèmes cloud publics, privés et hybrides changent la façon dont les organisations du monde entier, des startups avant-gardistes aux entreprises multinationales, envisagent leur infrastructure informatique.

La flexibilité et l'agilité, la rapidité de déploiement, la maîtrise des coûts, l'évolutivité, la sécurité renforcée et la liberté de localisation sont autant de caractéristiques de ce modèle.

#### 2.2.2. Les modèles de déploiement de cloud computing

Il existe trois catégories de modèles de déploiement de cloud computing :

- **Cloud privé**

Il est conçu spécifiquement pour gérer et héberger l'infrastructure interne ou externe d'une entreprise. Ce modèle permet aux entreprises de contrôler la gestion et la sécurité de leur infrastructure. Cela permet aux équipes informatiques de provisionner, allouer et fournir des ressources à la demande.

- **Cloud public**

Des ressources cloud à la demande qui fournissent tout ce dont vous avez besoin, des tests au déploiement à grande échelle, avec un excellent rapport prix/performances et une grande évolutivité. Une approche de cloud public aide les organisations à tous les niveaux à se moderniser, depuis les startups cherchant à déployer des solutions le plus rapidement possibles

jusqu'aux organisations mondiales nécessitant des ressources à la demande pour des projets et des applications spécifiques.

- **Cloud hybride**

Cette solution cloud offre le « meilleur des deux mondes », combinant le bare metal et le cloud computing, intelligemment mélangés et intégrés pour maximiser leurs avantages. Par exemple, les utilisateurs du cloud peuvent s'appuyer sur une flotte de serveurs dédiés pour alimenter les sites et les applications, tout en utilisant le cloud pour automatiser et rationaliser le stockage des données. Les possibilités sont presque infinies.

#### 2.2.3. Les modèles de cloud computing

Bien que les services de cloud computing continuent d'évoluer, ils peuvent encore être divisés en trois catégories principales :

- **Infrastructure as a service (IaaS)**

En automatisant les éléments constitutifs d'une infrastructure de classe mondiale, les organisations à tous les niveaux peuvent maximiser le contrôle des coûts tout en bénéficiant d'une plus grande évolutivité et agilité. Sans avoir besoin de déployer, de gérer et de maintenir une infrastructure sur site, les organisations peuvent accroître leur liberté d'innover. Avec IaaS, les entreprises louent une infrastructure informatique (pour la puissance de calcul, le stockage ou la mise en réseau) auprès d'un fournisseur de services cloud, mais elles continuent de superviser la gestion de leurs applications critiques et de leurs systèmes, sécurité, bases de données et systèmes d'exploitation.

- **Platform as a service (PaaS)**

Avec PaaS, les équipes peuvent gérer, créer, tester et déployer leurs applications sur une plateforme cloud conçue pour profiter aux utilisateurs. L'infrastructure informatique sous-jacente, telle que le matériel et les middlewares, est gérée par un fournisseur cloud de confiance. Sans avoir besoin de maintenir l'infrastructure, les équipes informatiques internes peuvent se concentrer sur les besoins en matière de données et d'applications de l'entreprise, libérant ainsi du temps pour se concentrer sur la croissance continue de l'entreprise.

- **Software as a service (SaaS)**

Avec le modèle SaaS, la plateforme logicielle est hébergée dans un cloud externe. Les utilisateurs peuvent accéder au logiciel via Internet via un abonnement. Cela élimine le besoin

pour les organisations d'acheter, d'installer et de mettre à jour des plates-formes logicielles critiques tout en garantissant que les équipes du monde entier peuvent accéder à leurs outils critiques. Avec le SaaS, les utilisateurs n'ont plus qu'à se soucier de gérer leurs propres données au sein de l'application, tandis que le logiciel est géré dans le cloud par un fournisseur externe.

Ce diagramme (figure 8) résume les services cloud en localisant les responsabilités des fournisseurs et des utilisateurs pour chaque catégorie :

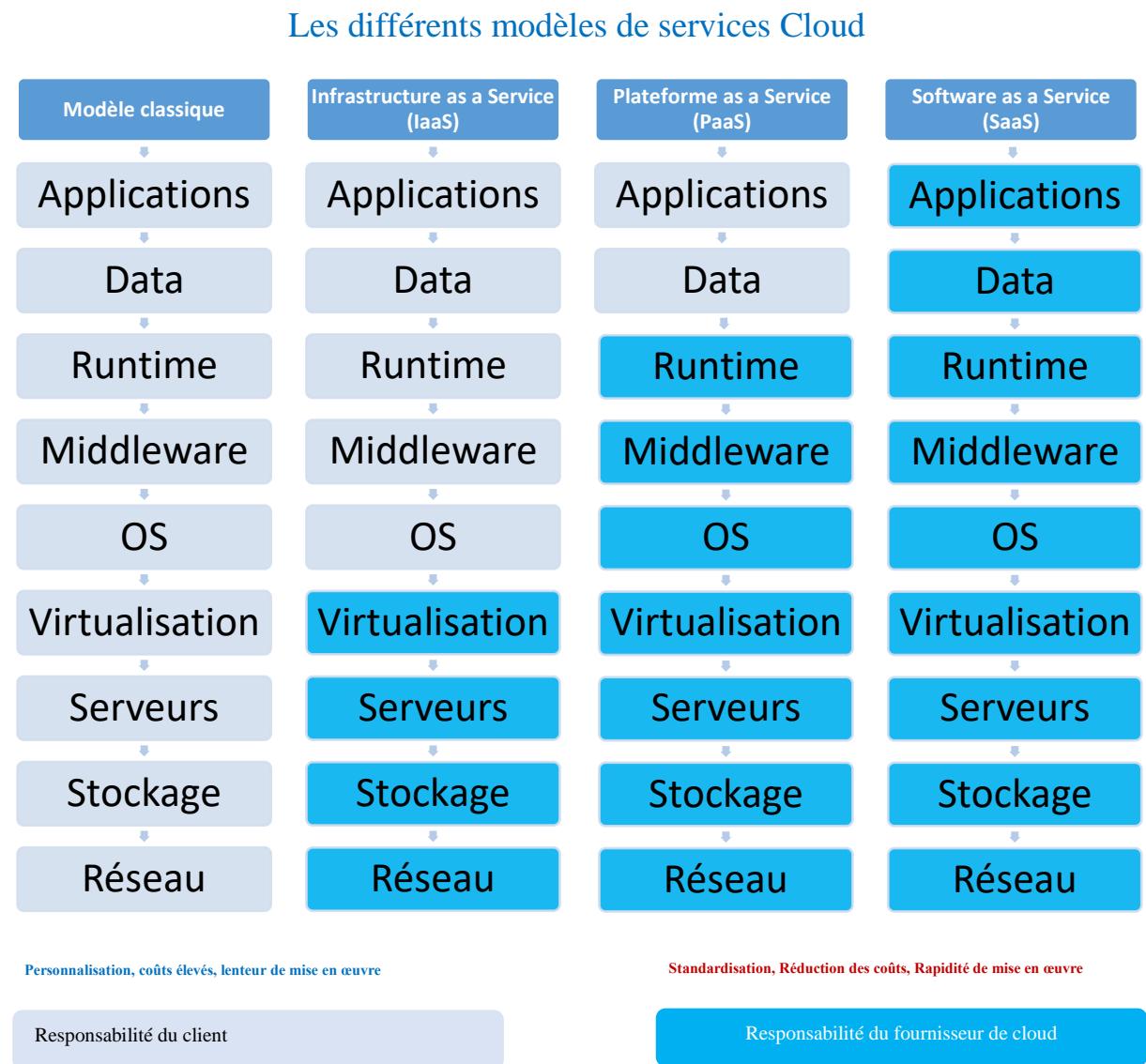


Figure 8 : Les différents modèles de services cloud

Lorsqu'elles choisissent le type de service qui répond à leurs besoins, les entreprises doivent savoir quoi faire et être capables d'identifier clairement (à leurs fournisseurs) ce qu'elles souhaitent faire.

#### 2.2.4. Avantages du cloud computing

Les entreprises bénéficient de ces différents services et de leurs nombreux avantages, ce qui leur permet de gagner du temps et de l'argent en réalisant des économies significatives.

- **Contrôle des coûts**

Avec un fournisseur de services cloud, vous ne payez que pour ce que vous utilisez et uniquement lorsque vous en avez besoin, de la technologie qui alimente vos produits et solutions à l'infrastructure informatique que votre organisation gère, prenant en charge le stockage quotidien ou à long terme des données sensibles.

- **Évolutivité**

Au fur et à mesure que votre entreprise se développe, cette évolutivité vous permet d'ajouter ou de supprimer des ressources et de la puissance. Cela donne aux entreprises plus de liberté pour se concentrer sur la manière d'utiliser l'architecture plutôt que sur le processus de déploiement.

- **Vitesse de déploiement**

La transition vers le cloud est de plus en plus courte. Pour les utilisateurs du cloud, le processus depuis la conception jusqu'à la production n'a jamais été aussi rapide et simple. Vous pouvez facilement faire évoluer votre solution de cloud computing en déployant de nouvelles ressources en quelques clics seulement.

- **Liberté d'emplacement**

Lors de la conception d'une infrastructure, il est important de prendre en compte l'emplacement des données, non seulement pour améliorer les performances du site et des applications, mais également pour obtenir la certification et répondre aux exigences de conformité (y compris les réglementations telles que le Cloud Act et le RGPD). L'hébergement sécurisé des ressources dans un centre de données externe signifie qu'elles peuvent être plus proches de vos équipes et des utilisateurs finaux afin de réduire la pression de conformité.

- **Sécurité renforcée**

La sécurité est depuis longtemps une préoccupation majeure pour de nombreuses organisations réticentes à migrer partiellement ou totalement vers le cloud. Cependant, pour répondre directement à ces problèmes, le cloud computing a considérablement évolué au fil des années, offrant une gamme de solutions cloud sécurisées.

### 2.2.5. Les principaux fournisseurs

Alors que le marché du cloud continue de se développer, les grands géants du web souhaitent s'emparer d'une part de ce marché. Ils offrent donc une variété de services adaptés pour être plus efficaces sur le marché. Il existe deux catégories de fournisseurs de cloud : le cloud public et privés.

Nous présentons ci-dessous les trois principaux fournisseurs de nuages publics:

- **Amazon Web Service (AWS)**

Pionnier dans le domaine, AWS est le cloud le plus complet et le plus utilisé au monde, avec plus de 200 services complets disponibles dans les centres de données du monde entier. Des millions de clients, notamment les startups à la croissance la plus rapide, les plus grandes entreprises et les principales agences gouvernementales, utilisent AWS pour réduire les coûts, améliorer la sécurité, devenir plus agiles et innover plus rapidement.

- **Microsoft Azure**

Lancé en 2010, Azure occupe la deuxième place avec 19 % de part de marché. Il compte plus de 200 produits et services cloud dans différentes catégories, notamment IaaS, PaaS et SaaS.

- **Google Cloud Platform**

GCP est un ensemble de services de cloud computing fournis par Google qui s'exécutent sur la même infrastructure que celle utilisée par Google pour ses produits destinés aux utilisateurs finaux, tels que YouTube, Gmail, etc. Google Cloud Platform fournit une variété de services, notamment (Calcul, Réseau, Machine learning and AI, Big data processing).

Services				
Machine virtuelles	Instances	VMs	VM Instances	
Plateforme-as-a-service	Elastic Beanstalk	Cloud Services	App Engine	
Serverless Computing	Lambda	Azure Functions	Cloud Functions	
Docker Management	ECS	Container Service	Container Engine	
Kubernetes Management	EKS	Kubernetes Service	Kubernetes Service	
Object Storage	S3	Block Blob	Cloud Storage	
Archive Storage	Glacier	Archive Storage	Coldline	
File Storage	EFS	Azure Files	ZFS / Avere	
Global Content Delivery	CloudFront	Delivery Network	Cloud CDN	
Managed Data Warehouse	Redshift	SQL Warehouse	Big Query	

Figure 9 : Fournisseurs Cloud: GCP vs Azure vs AWS

***En termes de coûts, de flexibilité et d'agilité, le cloud computing est devenu un excellent environnement pour héberger l'infrastructure de projets.***

### 2.3. Intégration continue

L'intégration continue (CI) est une pratique de développement dans laquelle les développeurs intègrent fréquemment (idéalement plusieurs fois par jour) du code dans un référentiel partagé. Chaque intégration peut ensuite être vérifiée via des builds et des tests automatisés. Bien que les tests automatisés ne fassent pas strictement partie de l'IC, ils sont souvent implicites. L'un des principaux avantages de l'intégration régulière est qu'elle facilite la détection des erreurs et leur localisation plus facilement. Puisque chaque modification introduite est généralement petite, la modification spécifique qui a introduit le défaut peut être rapidement identifiée.

Au cours des dernières années, l'IC est devenu une pratique exemplaire pour le développement de logiciels et est guidé par un ensemble de principes clés. Parmi ceux-ci figurent le contrôle des révisions, l'automatisation de la construction et l'automatisation des tests. De plus, le déploiement continu et la livraison continue sont devenus des bonnes pratiques afin que votre application puisse être déployée à tout moment et que même la base de code principale puisse être automatiquement mise en production lorsque de nouvelles modifications surviennent.

Cela permet à votre équipe d'avancer rapidement tout en maintenant des normes de qualité élevées et automatiquement vérifiables. "*L'intégration continue n'élimine pas les bugs, mais elle facilite leur recherche et leur suppression.*" [3]

## 2.4. Livraison continue

La livraison continue est une extension de l'intégration continue puisqu'elle déploie automatiquement toutes les modifications du code dans un environnement de test et/ou de production après l'étape de construction.

Cela signifie qu'en plus des tests automatisés, vous disposez d'un processus de publication automatisé et que vous pouvez déployer votre application à tout moment en cliquant sur un bouton.

En théorie, avec la livraison continue, vous pouvez décider d'une livraison quotidienne, hebdomadaire, bimensuelle, ou de tout ce qui correspond aux besoins de votre entreprise. Cependant, si vous voulez vraiment bénéficier des avantages de la livraison continue, vous devez déployer votre application en production le plus tôt possible afin de vous assurer que vous publiez de petits lots qui sont faciles à dépanner en cas de problème. [4]

## 2.5. Déploiement continu

Le déploiement continu va plus loin que la livraison continue. Avec cette pratique, chaque changement qui passe toutes les étapes de votre pipeline de production est mis à la disposition de vos clients. Il n'y a pas d'intervention humaine, et seul un test raté peut empêcher le déploiement d'une nouvelle modification en production.

Le déploiement continu est un excellent moyen d'accélérer la boucle de rétroaction avec vos clients et de soulager l'équipe, car il n'y a plus de "jour de publication". Les développeurs peuvent se concentrer sur la création de logiciels, et ils voient leur travail mis en production quelques minutes après l'avoir terminé. [5]

## 2.6. Éléments clés

Dans cette section, nous allons définir les éléments suivants : kubernetes, pod, docker, cluster, Kustomize, Runner, HELM, Portainer, Ingress, Cert-Manager.

### 2.6.1. Kubernetes

Kubernetes, ou K8S, est une solution d'orchestration de conteneurs. Plus précisément, il s'agit d'un logiciel d'orchestration qui permet le déploiement d'applications à l'aide de scripts de type YAML, la mise à l'échelle automatique des applications et de multiples possibilités de déploiement dans différents environnements.



Figure 10 : Logo Kubernetes

Kubernetes a été initialement développé et conçu par les ingénieurs de Google dans le cadre du projet Borg et donné à la Cloud Native Computing Foundation (CNCF) en 2015. [6]

#### 2.6.2. Pod

Un pod Kubernetes est une collection d'un ou plusieurs conteneurs Linux. La plus petite unité dans les applications Kubernetes. Un Pod peut contenir plusieurs conteneurs étroitement liés (applications avancées) ou un seul conteneur (applications plus courantes). [7]

#### 2.6.3. Docker

Docker a largement contribué à la démocratisation de la conteneurisation avec le lancement de sa plateforme de conteneurs en 2013. Il simplifie la création de conteneurs et d'applications basées sur des conteneurs.

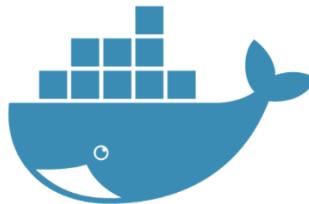


Figure 11 : Logo Docker

Il en existe d'autres, mais c'est le plus utilisé. Il est également plus facile à déployer et à utiliser que ses concurrents. [8]

#### 2.6.4. Container

Alternative aux méthodes de virtualisation traditionnelles basées sur des machines virtuelles, il s'agit d'un environnement d'exécution léger. L'une des pratiques les plus importantes dans le développement de logiciels modernes est l'isolation des applications déployées sur le même hôte ou sur un cluster. Cela les empêche d'intervenir. [8]

#### 2.6.5. Cluster

Dans les systèmes informatiques, un cluster est défini comme une grappe de serveurs au sein d'un réseau, les clusters de serveurs offrent de nombreux avantages en informatique. Il offre une disponibilité totale, une répartition de la charge et des capacités de calcul parallèle.

### 2.6.6. Kustomize

Kustomize est un outil Kubernetes qui vous permet de personnaliser le fichier YAML d'origine de la ressource k8s d'origine à des fins multiples (par exemple : différents environnements, différentes variables/répliques/ressources de calcul, etc.), en gardant le fichier YAML d'origine intact et disponible tel quel. [9]

**Kustomize** est un outil de gestion de configuration intégré dans Kubernetes!

### 2.6.7. Runner

GitLab Runner est un agent installé sur le serveur cible qui exécute des tâches et envoie les résultats à GitLab. Il fonctionne en conjonction avec le service d'intégration continue GitLab CI de GitLab.



Figure 12 : Logo Gitlab Runner

### 2.6.8. HELM

Helm est le gestionnaire de packages pour Kubernetes. Il permet d'installer des applications de n'importe quelle source sur le cluster grâce à deux commandes, le seul prérequis pour cette installation est la présence du binaire « helm » sur la machine.



Figure 13 : Logo HELM

### 2.6.9. Portainer

La gestion multi-cluster de Kubernetes est puissante mais complexe. L'interface graphique intuitive de Portainer permet aux administrateurs de gérer et de sécuriser le cluster rapidement, tandis que les valeurs par défaut intégrées et saines maintiennent l'équipe sur la bonne voie. Il offre également les fonctionnalités suivantes: RBAC pour Kubernetes,

approvisionnement des clusters KaaS, automatisation GitOps, importation via le fichier kubeconfig, gestion du registre, contraintes de sécurité des pods, authentication externe.

### 2.6.10. Ingress

Un Ingress est un objet Kubernetes utilisé pour gérer l'accès externe aux services d'un cluster, généralement le trafic HTTP. Ingress peut fournir un équilibrage de charge, une terminaison TLS et un hébergement virtuel basé sur le nom. [10]

### 2.6.11. Cert-Manager

Cert-Manager est un contrôleur pour la gestion des certificats. Les gestionnaires de certificats peuvent aider à émettre des certificats provenant de différents émetteurs tels que Let's Encrypt, HashiCorp Vault, Venafi, de simples paires de clés signées ou des certificats auto-signés. Cert-Manager vérifie les certificats, s'assure qu'ils sont à jour et les met à jour avant leur expiration. [11]

## 2.7. Étude comparative

Afin de mettre en œuvre un pipeline CI/CD ainsi qu'un tableau de monitoring qui rend compte de l'état du système, un ensemble d'outils est nécessaire : un serveur pour la gestion de versions du code, une plateforme d'orchestration et un serveur d'intégration et déploiement continu (CI/CD). En outre, pour faciliter la mise en place de certains de ces éléments, nous avons également besoin d'un outil de configuration et de provisionnement.

Nous présentons ci-dessous l'étude réalisée sur différents outils et plates-formes, qui se terminent par l'extraction des avantages et des inconvénients de chacun d'entre eux.

### 2.7.1. Gestion de versions : Gitlab vs SVN

Nous examinons deux des outils les plus populaires pour la gestion des versions : Gitlab et SVN, nous les avons comparés sur la base des éléments suivants : architecture, Single point of failure, contrôle d'accès, stockage de contenu, branche et espace mémoire utilisé.

Le tableau 2 fournit plus de détails sur la comparaison.

Aspect	GITLAB	SVN
Architecture	Dans un environnement distribué, chaque développeur peut avoir son propre clone sur sa machine en récupérant le code d'un dépôt Git central. Même s'il n'y a pas de connexion, les développeurs peuvent	Centralisé, l'historique complet des modifications n'est disponible que dans le répertoire central. Pour obtenir l'historique, les utilisateurs doivent communiquer avec l'annuaire central via le réseau. Les sauvegardes

	toujours créer une nouvelle branche, livrer un fichier et réviser une version.	sont gérées et maintenues indépendamment du SVN.
Single point of failure	Comme il y a autant de copies du fichier qu'il y a d'utilisateurs, voire plus, il n'y a pas de « single point of failure ».	Seules les données enregistrées dans le backup le plus récente peuvent être restaurées si le répertoire central est corrompu.
Contrôle d'accès	Le système étant distribué, il n'est pas nécessaire de disposer d'un « commit access ». Il vous suffit de choisir la source et la destination de la fusion.	Un « commit access » est requis en raison de la centralisation.
Stockage de contenu	Tout le contenu est stocké dans un dossier .git, qui est l'endroit où le code cloné est stocké sur la machine du client.	Sauvegarder les métadonnées des fichiers dans un dossier .svn caché
Les branches	Une branche dans Git n'est qu'un pointeur léger et déplaçable vers l'un de ces commits. La branche par défaut dans Git s'appelle master. Au fur et à mesure que la vérification progresse, la branche master pointe vers le dernier commit. Le pointeur de branche principale avance automatiquement à chaque validation.	Les branches ne sont qu'un autre dossier dans le référentiel de code et n'ont pas d'historique. Pour savoir si une branche a été fusionnée, vous devez exécuter une commande explicite. Ce type de gestion augmente la probabilité de création de branches « orphelines ». Il n'y a aucun moyen de savoir ce qui a changé une fois la fusion terminée.
Espace	Dépôt de code relativement petit (30x plus petit que SVN). En effet, le répertoire de travail de gitlab ne nécessite qu'un fichier d'indexation de 100 bits pour chaque fichier.	L'espace requis est relativement important. Puisque tous fichiers du répertoire de travail sont dupliqués, l'une pour travailler et l'autre cachée dans .svn.

Tableau 2 - Tableau de comparaison Gitlab vs SVN

Nous avons choisi Gitlab Server, qui répondait aux exigences fonctionnelles de la solution grâce à son architecture décentralisée et au gain de place qu'elle offrait (les dépôts sont 30 fois plus petits que SVN).

### 2.7.2. Outils d'orchestration : GitLab CI vs Jenkins

GitLab CI et Jenkins sont les plus grands noms de l'intégration et déploiement continu (CI/CD). CI/CD est un ensemble de processus de développement logiciel automatisés.

Jenkins intègre nativement le serveur CI/CD Jenkins, Kubernetes, Helm et d'autres outils pour offrir un pipeline CI/CD avec les meilleures pratiques intégrées, telles que l'utilisation de GitOps pour gérer les environnements.

GitLab CI/CD est un outil puissant intégré à GitLab qui vous permet d'appliquer toutes les pratiques de l'intégration et déploiement continu (CI/CD) à votre logiciel sans avoir besoin d'applications ou d'intégrations tierces. D'autre part, GitLab offre déjà plus que ce que Jenkins espère faire évoluer en fournissant une application unique et entièrement intégrée pour l'ensemble du cycle de vie DevOps.

Le tableau 3 compare ces deux outils, la comparaison se base sur les éléments suivants: l'aspect Open source du code, la facilité d'utilisation, nombre des plugins, suivi des problèmes, surveillance.

Aspect	Gitlab CI	Jenkins
Open source	Oui	Oui
Facilité d'utilisation	Oui	Non
Plugins	Oui	Oui
Suivi des problèmes	Oui	Non
Surveillance	Oui	Non

Tableau 3 - Tableau de comparaison Gitlab CI vs Jenkins

Outre les objectifs de Jenkins, GitLab fournit également des services de planification, de SCM (git), d'empaquetage, de publication, de configuration et de surveillance (en plus du CI/CD sur lequel Jenkins se concentre).

De plus, les fonctionnalités natives de Jenkins peuvent être étendues via des plugins. Leur maintenance, leur protection et leur mise à jour sont coûteuses. En revanche, GitLab est un noyau ouvert, n'importe qui peut apporter des modifications directement à la base de code, et une fois la base de code fusionnée, chaque modification est automatiquement testée et maintenue.

De plus, GitLab est également un véritable outil de collaboration et de gestion de projet (agile), vous n'avez donc pas besoin d'intégrer d'autres outils de collaboration (par exemple Jira).

Enfin, nous avons choisi la plateforme open source GitLab car elle offre un large éventail de fonctionnalités qui nous permettent de centraliser, sécuriser et automatiser plusieurs étapes du processus DevSecOps.

*GitLab une seule application pour tout le cycle de vie devops !*

### 2.7.3. Virtualisation : Docker vs Machine virtuelle

Docker (Conteneur) et les machines virtuelles (VM) sont deux technologies de déploiement d'applications. Dans le cycle de vie du développement logiciel, le déploiement prépare le code d'application que les utilisateurs finaux peuvent exécuter. Docker est une plate-forme open source que les développeurs utilisent pour regrouper des logiciels dans des unités standardisées appelées conteneurs. Un conteneur contient le code d'application et son environnement, y compris les bibliothèques, les outils système et l'exécution. Docker vous permet de déployer et de faire évoluer des applications sur n'importe quelle machine et d'assurer une exécution cohérente de votre code.

Une machine virtuelle, en revanche, est une copie numérique d'une machine physique. Vous pouvez exécuter plusieurs machines virtuelles avec leurs propres systèmes d'exploitation sur le même système d'exploitation hôte. Les développeurs configurent les machines virtuelles pour créer des environnements d'application. Vous pouvez également exécuter des conteneurs Docker sur des machines virtuelles. [12]

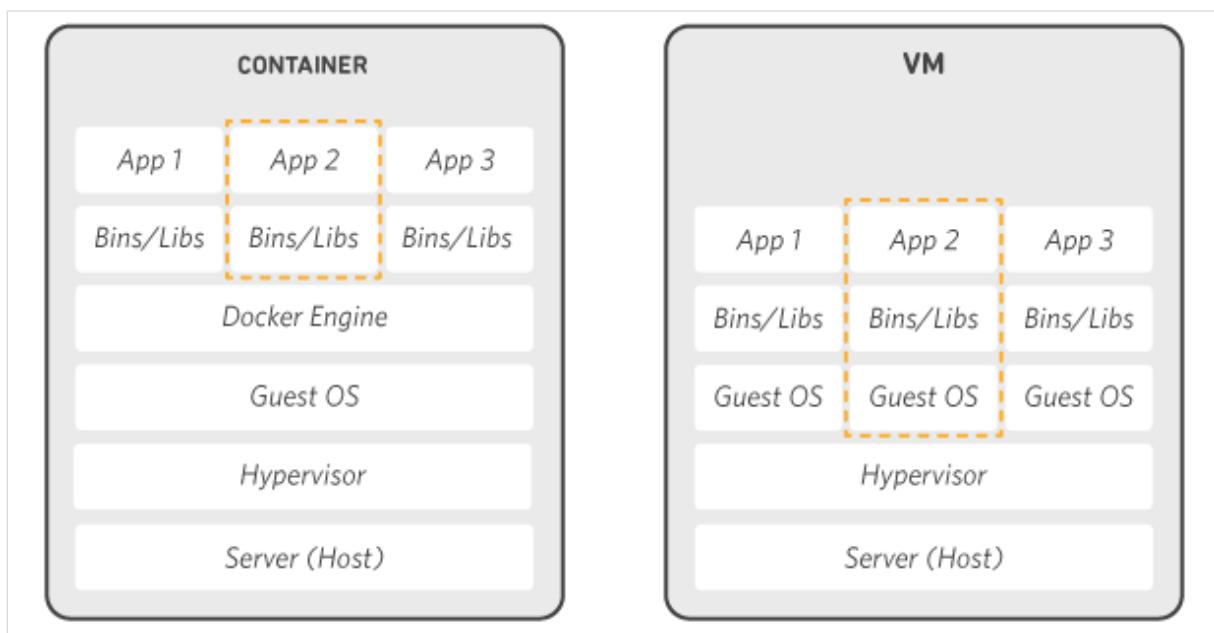


Figure 14 : Container vs VM

### 2.7.4. Plateforme de déploiement : Kubernetes vs Docker Swarm

Kubernetes et Docker Swarm sont deux plateformes d'orchestration open-source de conteneurs qui aident les développeurs à créer, déployer et gérer des applications et des services sur différents modèles du cloud, tout en maintenant la portabilité entre différents fournisseurs d'infrastructure.

Ces plateformes populaires d'orchestration de conteneurs partagent certaines similitudes en termes d'objectifs et de fonctionnalités, notamment :

1. Orchestration de conteneurs
2. Équilibrage de charge
3. Évolutivité
4. Découverte de services et mise en réseau
5. Auto-régénération et haute disponibilité
6. Rolling Updates et Rollbacks
7. Gestion du cycle de vie des conteneurs
8. Portabilité

Nous sommes très intéressés par ces outils en raison de leur capacité à être intégrés dans une chaîne DevOps.

Aspect	Kubernetes	Docker Swarm
Simplicité	Non	Oui
Installation	<ul style="list-style-type: none"> <li>– Installation complexe</li> <li>– Configuration : plus d'effort manuel.</li> </ul>	<ul style="list-style-type: none"> <li>– Simple à installer</li> <li>– Simple à configurer</li> </ul>
Déploiement d'applications	<p>Offre une gamme plus large d'objets et d'options</p> <ul style="list-style-type: none"> <li>– Déploiements</li> <li>– Pods</li> <li>– Namespaces</li> </ul>	<ul style="list-style-type: none"> <li>– Stack</li> <li>– Services</li> <li>– Microservice</li> </ul>
Fonctionnalités	Plusieurs fonctionnalités	Ensemble limité de fonctionnalités
Écosystème	L'écosystème et la communauté sont vastes	L'écosystème et la communauté sont plus petits
Limites de la mise à l'échelle	Capacités plus avancées de mise à l'échelle et de distribution de la charge de travail	Capacité limitée pour les déploiements à très grande échelle ou les charges de travail très dynamiques
Surveillance et observabilité	une surveillance intégrée et un large éventail d'intégrations disponibles	Basique, peuvent avoir besoin d'investir dans des outils de surveillance et de journalisation supplémentaires
Maturité et stabilité	largement adopté par les organisations de toutes tailles, y compris les grandes entreprises, les	Considéré comme moins mature et moins stable

	startups et les leaders technologiques	
Ordonnancement et placement des conteneurs	L'algorithme de planification avancé et fin	Dans certains scénarios, les entreprises peuvent avoir besoin d'un contrôle plus précis
Intégrations	un large éventail d'intégrations avec des tiers	Présenter des limites lorsqu'il s'agit d'intégrer des outils ou des services tiers

Tableau 4 - Tableau de comparaison Kubernetes vs Docker Swarm

Nous avons choisi kubernetes pour sa maturité, sa stabilité, ses performances et sa capacité à s'intégrer avec différentes plateformes tierces.

#### 2.7.5. Supervision : Prometheus & Grafana vs Zabbix

Pour surveiller notre infrastructure, nous avions le choix entre Prometheus & Grafana ou Zabbix, qui sont tous deux des outils modernes de surveillance automatisée.

Ces outils peuvent aider les SRE (Software reliability Engineer) et les praticiens DevOps à analyser la santé de l'environnement de production dans des graphiques en temps réel.

La comparaison de ces deux outils dans le tableau 2 est basée sur : Node Exporter, support plateforme, découverte de services, flexibilité et communication

Aspect	Prometheus & Grafana	Zabbix
Node Exporter	Support	Ne supporte pas
Support plateforme	Native support de K8S et Docker	Ne supporte pas K8S et Docker
Découverte de services	Découverte basé sur les services	Découverte basé sur les machines
Flexibilité	Déclarative config YAML (active, désactive) pour Node Exporter	Ne supporte pas la configuration déclarative.
Communication	Communication basée sur HTTP, ce qui rend l'intégration avec d'autres outils beaucoup plus facile	Communication basée sur TCP

Tableau 5 - Tableau de comparaison Prometheus & Grafana vs Zabbix

Pour garantir une plus grande flexibilité, une meilleure intégration et la puissance d'un plugin tiers, nous avons décidé d'utiliser Prometheus et Grafana.

### 2.7.6. SonarQube vs CodeScan

Le serveur de qualité de code présente un certain nombre d'avantages pour tout projet : Il offre une analyse approfondie de la qualité d'une application à l'aide d'un ensemble complet de statistiques. L'analyse de la qualité du code permet de suivre son évolution dans le temps.

Le tableau 6 présente une comparaison de ces deux outils de d'analyse de code :

Aspect	SonarQube	CodeScan
Open source	Oui	Non
Déploiement	<ul style="list-style-type: none"> <li>– Basé sur le cloud</li> <li>– On-Premise</li> </ul>	<ul style="list-style-type: none"> <li>– Basé sur le cloud</li> </ul>
Langage supportés	27+	25+

Tableau 6 - Tableau de comparaison CodeScan vs SonarQube

Après comparaison, nous avons décidé d'utiliser SonarQube comme serveur de qualité de code.

### 2.7.7. Outil de configuration et d'approvisionnement : Terraform vs Ansible

Configurer un cluster AKS (Azure Kubernetes Service) à l'aide de l'outil « Azure CLI » propre à Azure est simple et rapide, mais ne nous permet pas d'être plus précis et granulaire lors de la définition des groupes de ressources, par exemple. Pour cela, nous avons choisi de travailler avec l'un des outils de configuration et de provisionnement Terraform et Ansible.

Terraform est un outil conçu pour faciliter le provisionnement et le dé provisionnement de l'infrastructure en cloud en utilisant une approche « Infrastructure as Code ». Il est très spécialisé dans ce domaine. En revanche, Ansible est un outil plus général qui peut être utilisé pour l'automatisation dans divers domaines. Terraform (*jusqu'à la version 1.5.6*) et Ansible disposent tous deux de solides communautés open-source et de produits commerciaux bien soutenus.

Ci-dessous un Tableau de comparaison entre ces deux derniers outils [13]:

Terraform	Ansible
Terraform est un outil d'approvisionnement.	C'est un outil d'approvisionnement cloud et de gestion des configurations.
Il suit une approche déclarative de l'infrastructure en tant que code.	Il suit une approche à la fois déclarative et procédurale.
Il est le mieux adapté à l'orchestration de services en cloud et à la mise en place d'une infrastructure en cloud à partir de zéro.	Il est principalement utilisé pour configurer les serveurs avec les bons logiciels et mettre à jour les ressources déjà configurées.

Par défaut, Terraform ne prend pas en charge l'approvisionnement en métal nu.	Ansible prend en charge l'approvisionnement des serveurs en métal nu.
Il n'offre pas de meilleur support en termes de packaging et de templating.	Il fournit un support complet pour le packaging et le templating.
Il dépend fortement de la gestion du cycle de vie ou de l'état.	Il ne dispose pas du tout de la gestion du cycle de vie.

Tableau 7 - Tableau de comparaison Terraform vs Ansible

Nous avons choisi Terraform parce qu'il est mieux adapté à l'orchestration des services cloud et à la création d'une infrastructure cloud à partir de zéro.

## 2.8. Synthèse et choix

Après avoir réalisé l'étude comparative, voici un résumé des technologies sélectionnées. Nous avons choisi Gitlab comme serveur de contrôle de version et Gitlab CI comme outil d'orchestration. Nous avons choisi d'utiliser AKS (Azure Kubernetes Service) pour le déploiement et l'hébergement de notre application. Enfin, nous avons utilisé Prometheus & Grafana pour la supervision.

Le tableau 8 résume notre choix de technologies.

Fonctionnalité	Outil
Serveur gestion de version	Gitlab
Serveur d'orchestration	Gitlab CI
Plateforme de déploiement continu	Kubernetes AKS
Supervision	Prometheus & Grafana
Serveur de qualité du code	SonarQube

Tableau 8 - les choix technologiques

## 2.9. Conclusion

Nous introduisons dans ce chapitre, les notions d'intégration et de déploiement continu, puis introduisons les outils utilisés aux différentes étapes du projet, puis explorons une étude comparative des plateformes utilisées, et enfin résumons les choix technologiques.

# Chapitre 3

## ANALYSE ET SPÉCIFICATIONS

### 3.1. Introduction

Pour déterminer les fonctionnalités que nous devons mettre en œuvre dans notre système, nous commençons par l'identification des acteurs de chaque sous-système. Nous définissons ensuite les exigences fonctionnelles, non fonctionnelles et techniques du projet dans le cadre d'une analyse détaillée. Enfin, nous avons représenté nos exigences en UML en créant et en rédigeant des diagrammes de cas d'utilisation en conjonction avec des descriptions textuelles.

### 3.2. Détermination des acteurs

Nous avons identifié 4 acteurs appartenant à l'équipe de développement et d'administration:

- **Client** : Un client doit avoir un accès sécurisé à son instance d'application.
- **Développeur** : c'est l'utilisateur de GITLAB, il peut stocker ou récupérer du code depuis le référentiel, Il est capable de consulter, de surveiller et de déclencher le pipeline CI/CD. Il joue également le rôle de testeur après chaque nouveau déploiement.
- **Chef de projet** : est responsable de la gestion du projet, de l'affectation des développeurs et peut accéder à des tableaux de bord de suivi, qui lui permettent de voir l'état du cluster et de suivre l'ensemble du cycle de développement de l'application sur Gitlab.
- **Administrateur** : Il est responsable de la mise en place des pipelines et participe à la phase de configuration du système. Il est également responsable du suivi de toutes les plateformes.

Dans certaines phases de l'exécution de notre système, des composants agissent comme des acteurs:

- **GITLAB** : GitLab est un référentiel Git basé sur le Web qui fournit des référentiels ouverts et privés gratuits, des fonctionnalités de suivi des problèmes et des wikis.
- **GITLAB-CI** : Il s'agit d'un orchestrateur et d'un outil d'intégration continue essentiel pour déclencher des builds.
- **Runner** : C'est un agent installé sur le serveur cible qui exécute des tâches et envoie les résultats à GitLab. Il fonctionne en conjonction avec GitLab CI.
- **Kaniko** : C'est l'outil utilisé pour construire de manière sécurisée l'image du conteneur sans utiliser Docker, et fait partie de notre processus de déploiement. Une fois construite, l'image est soumise au registre de containers Gitlab.

- **Registre Gitlab** : c'est le registre de containers qui est complètement intégré à Gitlab et permet de stocker nos images docker.
- **SonarQube** : l'outil qui nous permettra d'analyser le code source de notre projet pendant l'exécution de notre pipeline.

### 3.3. Analyse des besoins

L'analyse des besoins pour notre système se compose de trois sections, les exigences fonctionnelles, les exigences techniques et enfin les exigences non fonctionnelles à prendre en compte.

#### 3.3.1. Détermination des exigences fonctionnelles

Nous listons ci-dessous les opérations fonctionnelles que les acteurs du système peuvent effectuer :

- Les développeurs peuvent récupérer et publier du code dans un référentiel Git.
- Les administrateurs peuvent ajouter et modifier des scripts de pipeline.
- Les administrateurs peuvent créer des templates pour créer et déployer des applications sur Kubernetes.
- Déclenchez automatiquement les builds sur Gitlab après avoir déposé le code dans un référentiel Git.
- Consulter le build sur le registre de Gitlab.
- Déployer automatiquement des applications sur Kubernetes.
- Le système garantit un accès sécurisé à l'instance d'application.
- Après la phase de livraison continue, les testeurs peuvent accéder à l'instance d'application dédiée pour tester les fonctionnalités développées.

Le chef de projet est autorisé à surveiller les ressources du système et le pipeline :

- Surveiller l'utilisation de CPU, l'utilisation de la RAM et la pression exercée sur le réseau par les pods.
- Le statut des différentes applications sur Portainer (conteneur, créé, en cours d'exécution, en attente, etc.) peut être consulté par le chef de projet.
- Les chefs de projet peuvent afficher l'historique et l'état actuel des pipelines à travers les projets et les groupes dans un seul tableau de bord.

#### 3.3.2. Détermination des exigences techniques

La liste ci-dessous exprime nos exigences techniques :

- Compte sur Gitlab, leur fonctionnalités référentiel Git, Gitlab CI, registre de conteneurs, service d'authentification, suivi des tâches et des projets, collaborations.
- Compte sur Microsoft Azure avec accès à Azure Kubernetes Service (AKS)
- Outils de déploiement et configuration : HELM, Terraform, Kustomize
- Services préconfiguré sur nos clusters AKS (Portainer, Ingress-Controller, Cert-Manager, PostgreSQL, PGAdmin, Prometheus, Grafana, Gitlab Agent, Gitlab Runner, SonarQube)

### 3.3.3. Les exigences non fonctionnelles

Pour que notre système soit robuste face aux défis de la charge de travail, à l'évolutivité inattendue et aux incidents, certains critères doivent être respectés, notamment les suivants :

- **Évolutivité** : Afin que nous puissions ajouter autant de nœuds au cluster Kubernetes que nécessaire, le système doit être capable de prendre en charge l'évolution et la scalabilité de ses composants.
- **Performance** : La qualité du système, les processus de pipeline et les tableaux de bord de surveillance doivent avoir une réponse instantanée.
- **Convivialité** : Le dashboard de monitoring doit être intuitif, c'est-à-dire disposer d'une interface graphique facile à comprendre qui facilite l'interaction entre l'utilisateur et le logiciel.

### 3.4. Représentation des exigences

Dans cette section, nous transformons les exigences précédemment identifiées en diagrammes de cas d'utilisation.

Avec un diagramme général des cas d'utilisation, on peut présenter de manière semi-formelle, les principaux sous-systèmes et les fonctions que le système doit remplir. Nous avons identifié plusieurs sous-systèmes pour le pipeline, à savoir :

- Intégration Continue ou Continuous Integration (CI).
- Déploiement Continu ou Continuous Deployment (CD).
- Configuration.
- Supervision.
- Pré-configuration.

Dans la figure 15, les cas d'utilisation de notre système explorent tous les acteurs et les sous-systèmes dans un diagramme général.

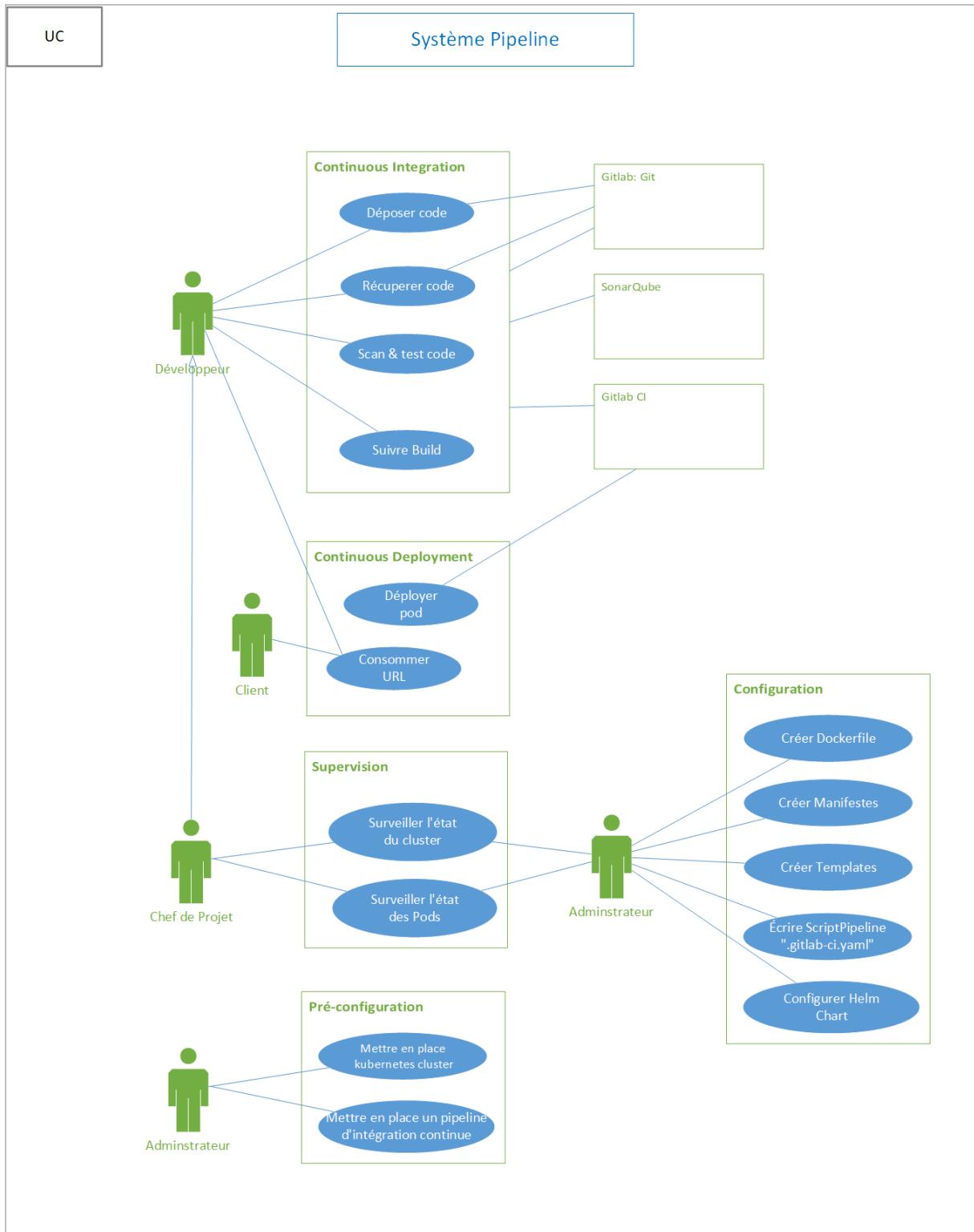


Figure 15 : Diagramme général du système

Les principales fonctionnalités du système sont mises en évidence dans le diagramme général des cas d'utilisation. Pour notre projet, nous avons identifié cinq sous-systèmes: l'intégration continue, le déploiement continu, la supervision du système, pré-configuration et la configuration, qui consiste à mettre en œuvre le pipeline d'intégration.

Le développeur peut interagir avec le système de différentes manières. Il peut :

- Suivre le build et le déploiement sur GITLAB CI.
- Déposer ou récupérer du code depuis GITLAB.
- Consulter l'état et le log des pods et services sur Portainer.
- Tester l'application déployée sur Kubernetes.

Le développeur en charge du poste de chef de projet peut également gérer :

- Consulter le dashboard de monitoring, vérifier les ressources (CPU, RAM, l'E/S Réseau) du cluster AKS.
- Contrôler l'état des pods et des services existants pour chaque projet, en précisant quelles fonctionnalités sont disponibles et lesquelles ne le sont pas.

Les clients peuvent accéder à leur instance d'application de manière sécurisée via l'utilisation de l'URL exposé par Kubernetes.

L'administrateur peut mettre en place, configurer et provisionner un cluster AKS (Azure Kubernetes Service).

L'Administrateur sera chargé de définir des pipelines CI/CD innovants qui prennent en charge l'environnement et les technologies utilisées.

### 3.4.1. Intégration continue

La phase ou le processus d'intégration continue est la première partie de notre pipeline CI/CD, dont l'exécution dépend généralement d'un commit effectué par un développeur dans le référentiel Git. Par la suite, la détection d'une nouvelle version du code par Gitlab-CI lance les jobs définies pour l'analyse et le build d'une image. Nous décrivons ce scénario plus en détail ci-dessous dans la description du texte.

Nous expliquant les fonctionnalités de cette partie du système « Intégration Continue », via le diagramme de cas d'utilisation illustré dans la figure 16.

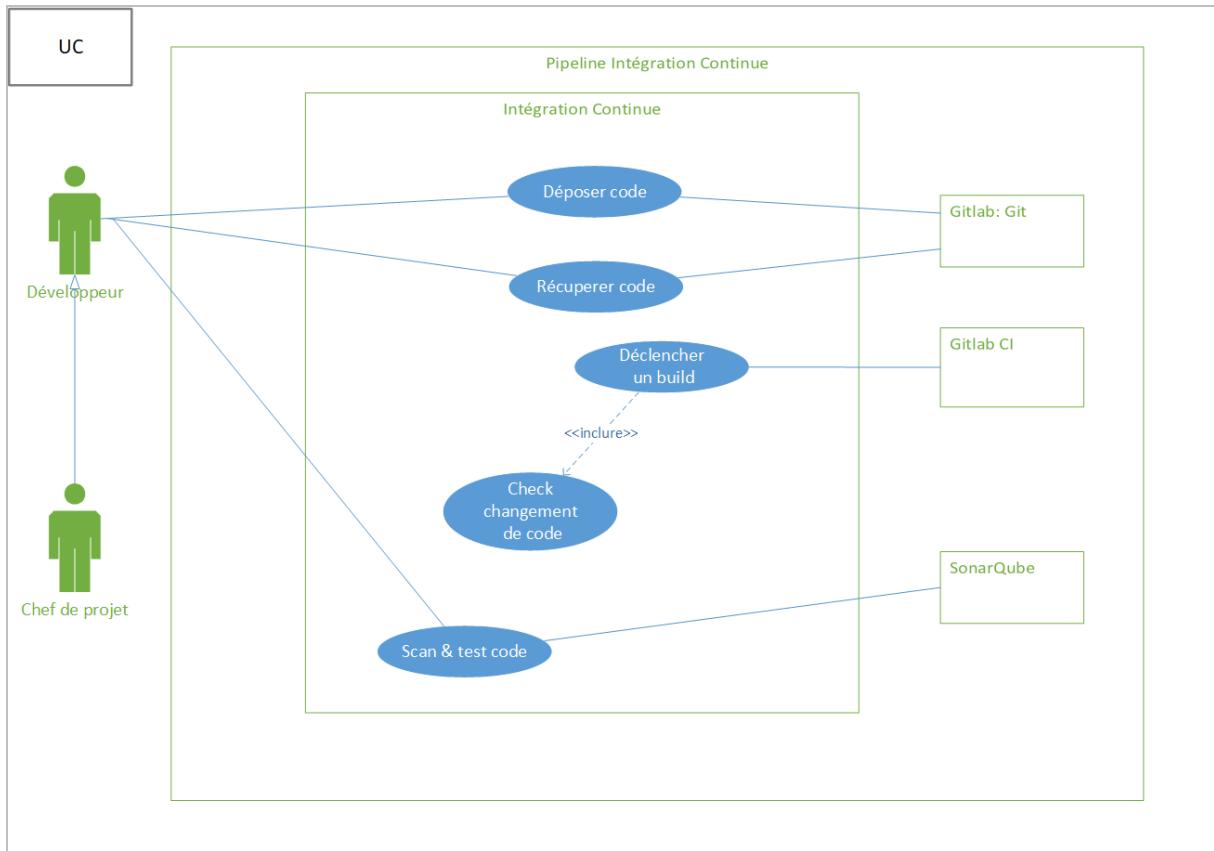


Figure 16 : Illustration du diagramme de partie "Intégration continue"

Nous décrivons le cas d'utilisation « Déposer code » avec une description textuelle dans le tableau 9.

Acteur :	Développeur
Précondition :	<ul style="list-style-type: none"> <li>– Serveur Gitlab accessible via ssh ou http</li> <li>– Le développeur a l'autorisation d'accéder à un projet</li> </ul>
Post condition :	Le développeur envoie son code sur le serveur Gitlab.
Description :	Le développeur devra effectuer un "push" pour envoyer son code sur le serveur GitLab.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Un push est effectué par le développeur</li> <li>2. le serveur Gitlab est mis à jour avec la dernière version du code.</li> </ol>
Scénario Alternatif :	<b>A1 : Détection des conflits</b> <ol style="list-style-type: none"> <li>1. Intervention manuelle pour résoudre le conflit</li> <li>2. Intervention automatique pour résoudre le conflit</li> </ol>
Scénario d'erreur :	Action non autorisée pour le développeur, en raison de droits d'accès manquants ou du fait que le serveur Gitlab n'est pas accessible.

Tableau 9 - Description textuelle du cas d'utilisation "Déposer code"

Nous décrivons le cas d'utilisation « Récupération du code » avec une description textuelle dans le tableau 10.

<b>Acteur :</b>	<b>Développeur</b>
Précondition :	<ul style="list-style-type: none"> <li>– Serveur Gitlab accessible via ssh ou http</li> <li>– Le développeur a l'autorisation d'accéder à un projet</li> </ul>
Post condition :	Récupération du code source par le développeur
Description :	Pour que le développeur puisse récupérer le code sur le serveur Gitlab, il doit exécuter la commande « git pull ».
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Exécution de la commande « git pull »</li> <li>2. Code cloné dans le répertoire local du développeur</li> </ol>
Scénario d'erreur :	Connexion non autorisée pour le développeur, en raison de droits d'accès manquants ou du fait que le serveur Gitlab n'est pas accessible.

Tableau 10 - Description textuelle du cas d'utilisation "Récupération du code"

<b>Acteur :</b>	<b>SonarQube</b>
Précondition :	Changement du code (.gitlab-ci.yml, extra_addons)
Post condition :	Déclenchement du scan
Description :	Le serveur CI de Gitlab déclenche automatiquement une analyse du code situé dans le référentiel Git sous « extra_addons ».
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Changements de code apportés par les développeurs.</li> <li>2. Gitlab CI détecte les modifications de code au niveau du référentiel Git dans Gitlab et déclenche des analyses.</li> <li>3. Un message est affiché indique que le job d'analyse se termine avec succès.</li> </ol>
Scénario Alternatif :	Gitlab CI n'a détecté aucune modification de code à l'étape 2. Après vérification des conditions définies par le développeur lors de l'exécution du job, le scénario nominal reprend au point 2.
Scénario d'erreur :	Lors d'une dépendance entre les jobs, et qu'un job s'exécute uniquement lorsque tous les travaux des étapes précédentes réussissent.

Tableau 11 - Description textuelle du cas d'utilisation "Déclencher un scan du code"

<b>Acteur :</b>	<b>Gitlab CI</b>
Précondition :	Changement du code (.gitlab-ci.yml, extra_addons)
Post condition :	Déclenchement de build
Description :	Un build est déclenché automatiquement par le serveur Gitlab-CI lorsqu'une modification du code est détectée dans le référentiel Git et qu'elle correspond aux conditions d'exécution du job build.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Nouvelle version du code est reçue dans le référentiel Git.</li> <li>2. Gitlab CI détecte une modification du code au niveau de référentiel Git dans Gitlab et déclenche un build.</li> <li>3. Un message de succès est affiché par Gitlab CI.</li> </ol>
Scénario Alternatif :	<b>A1 : Aucune modification du code</b> <ol style="list-style-type: none"> <li>1. Gitlab CI reste en mode recherche de modification avec vérification des conditions définie par le développeur dans l'exécution d'un job.</li> </ol>
Scénario d'erreur :	Lors d'une dépendance entre les jobs, et qu'un job s'exécute uniquement lorsque tous les travaux des étapes précédentes réussissent.

Tableau 12 - Description textuelle du cas d'utilisation "Déclencher un build"

### 3.4.2. Déploiement continu

Le processus de déploiement continu est la deuxième partie de notre pipeline CI/CD, son exécution se produit lorsque toutes les étapes de la première phase d'intégration continue se terminent avec succès et une fusion du code dans le référentiel Git de la branche de développement vers la branche de production, Gitlab-CI effectuera le déploiement, ou redéploiement, en mettant à jour l'image vers la nouvelle version créée.

Nous présentons tous les fonctionnalités de cette phase du pipeline avec le diagramme de cas d'utilisation cité dans la figure 17.

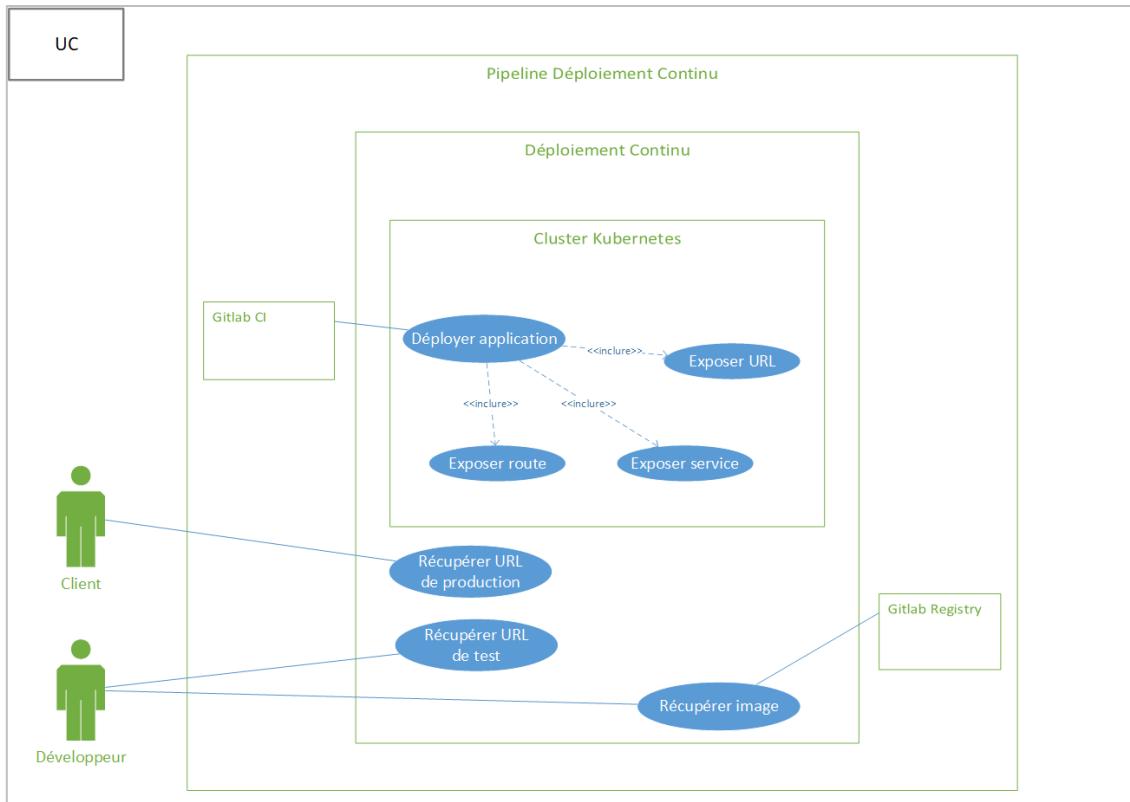


Figure 17 : Diagramme du cas d'utilisation "Déploiement continu"

<b>Acteur :</b>	<b>Gitlab CI</b>
Précondition :	<ul style="list-style-type: none"> <li>– Build se termine avec succès</li> <li>– L'image Docker est créée.</li> </ul>
Post condition :	<ul style="list-style-type: none"> <li>– Déploiement réussi de l'application</li> <li>– L'application est accessible via une route</li> </ul>
Description :	Le serveur Gitlab CI, vérifie périodiquement si un changement au niveau du code est détecté sur référentiel Git dans Gitlab, si des modifications correspondent aux conditions d'exécution, un build est automatiquement déclenché.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Au niveau d'un nœud de cluster, Kubernetes crée un pod.</li> <li>2. Le pod se voit attribuer une adresse IP interne, qui n'est accessible qu'aux nœuds du cluster.</li> <li>3. Avec l'utilisation de l'image Docker créée précédemment, kubernetes créer un conteneur à l'intérieur du pod.</li> <li>4. Le service du pod est exposé par Kubernetes.</li> <li>5. Une route vers l'application est définie en déployant un Ingress.</li> </ol>
Scénario d'erreur :	Le pod ne peut pas être créé car il n'y a pas assez d'espace dans les nœuds.

Tableau 13 - Description textuelle du cas d'utilisation "Déployer application"

Acteur :	Développeur
Précondition :	Une image existante dans le registre de Gitlab.
Post condition :	Image est prête à utiliser par le développeur.
Description :	Pour des tests en local, le développeur peut récupérer l'image à partir du registre Gitlab.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Exécution de la commande « docker pull image », en spécifiant l'adresse du registre GitLab.</li> <li>2. L'image est transférée au développeur couche par couche.</li> </ol>
Scénario Alternatif :	<b>A1 : login ou mot de passe incorrecte :</b> <ol style="list-style-type: none"> <li>1. Un message d'erreur est affiché par le système : « Echec de connexion ».</li> </ol>
Scénario d'erreur :	L'image ne peut pas être récupérée car l'adresse du registre GitLab spécifiée est incorrecte.

Tableau 14 - Description textuelle du cas d'utilisation "Récupérer image"

Acteur :	Client
Précondition :	<ul style="list-style-type: none"> <li>- Application testé et prête</li> <li>- Une route sécurisée est définie vers l'application du client.</li> </ul>
Post condition :	Application accessible par l'URL
Description :	Le client clique sur l'URL exposée par Kubernetes pour accéder à l'application dans leur navigateur ou application mobile.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Le client peut accéder à son instance d'application via l'URL fournie.</li> <li>2. Le client utilise les informations d'accès pour accéder via l'application mobile.</li> </ol>
Scénario Alternatif :	<b>A1 : login ou mot de passe non valide :</b> <ol style="list-style-type: none"> <li>1. Un message d'erreur est affiché par le système : « Votre login ou votre mot de passe est incorrect ».</li> </ol>
Scénario d'erreur :	Si aucune route n'est exposée sur le nœud où l'application est déployée, l'application ne sera pas disponible.

Tableau 15 - Description textuelle du cas d'utilisation "Récupérer l'URL de production"

Acteur :	Développeur (testeur)
Précondition :	<ul style="list-style-type: none"> <li>- Application prête</li> <li>- Une route non sécurisée est définie vers l'application.</li> </ul>
Post condition :	Instance de test est accessible.
Description :	Le testeur peut accéder à l'application via l'URL exposée par Kubernetes.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Le testeur utilise l'adresse URL afin d'accéder à l'application et vérifier les nouvelles fonctionnalités ajoutées.</li> </ol>

Scénario Alternatif :	<b>A1 : login ou mot de passe non valide :</b> 1. Un message d'erreur est affiche par le système : « Votre login ou votre mot de passe est incorrect ».
Scénario d'erreur :	Si aucune route n'est exposée sur le nœud où l'application est déployée, l'application n'est pas accessible.

Tableau 16 - Description textuelle du cas d'utilisation "Récupérer l'URL du test"

### 3.4.3. Configuration

Ce module consiste à développer des manifestes, des modèles et des fichiers de configuration, puis à les utiliser pour le build et le déploiement des applications soit à partir de la ligne de commande, soit par le biais d'appels au pipeline CI/CD. L'ajout continu de processus sous forme de job permet d'intégrer d'autres outils dans le pipeline.

La figure 18 présente un diagramme de cas d'utilisation illustrant la phase de configuration.

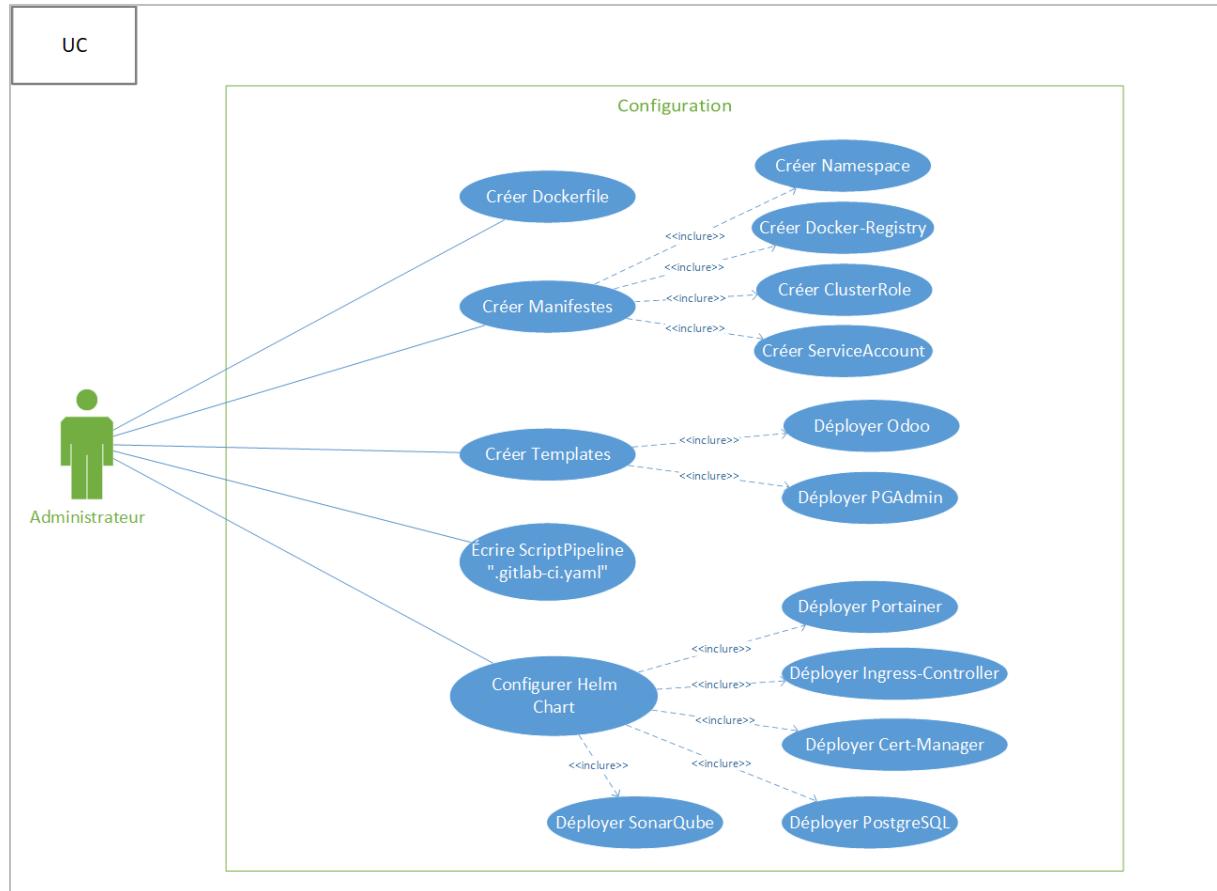


Figure 18 : Diagramme de cas d'utilisation "Configuration"

Dans la figure 19, nous présentons deux méthodes utilisées pour faire le déploiement sur la plateforme Kubernetes.

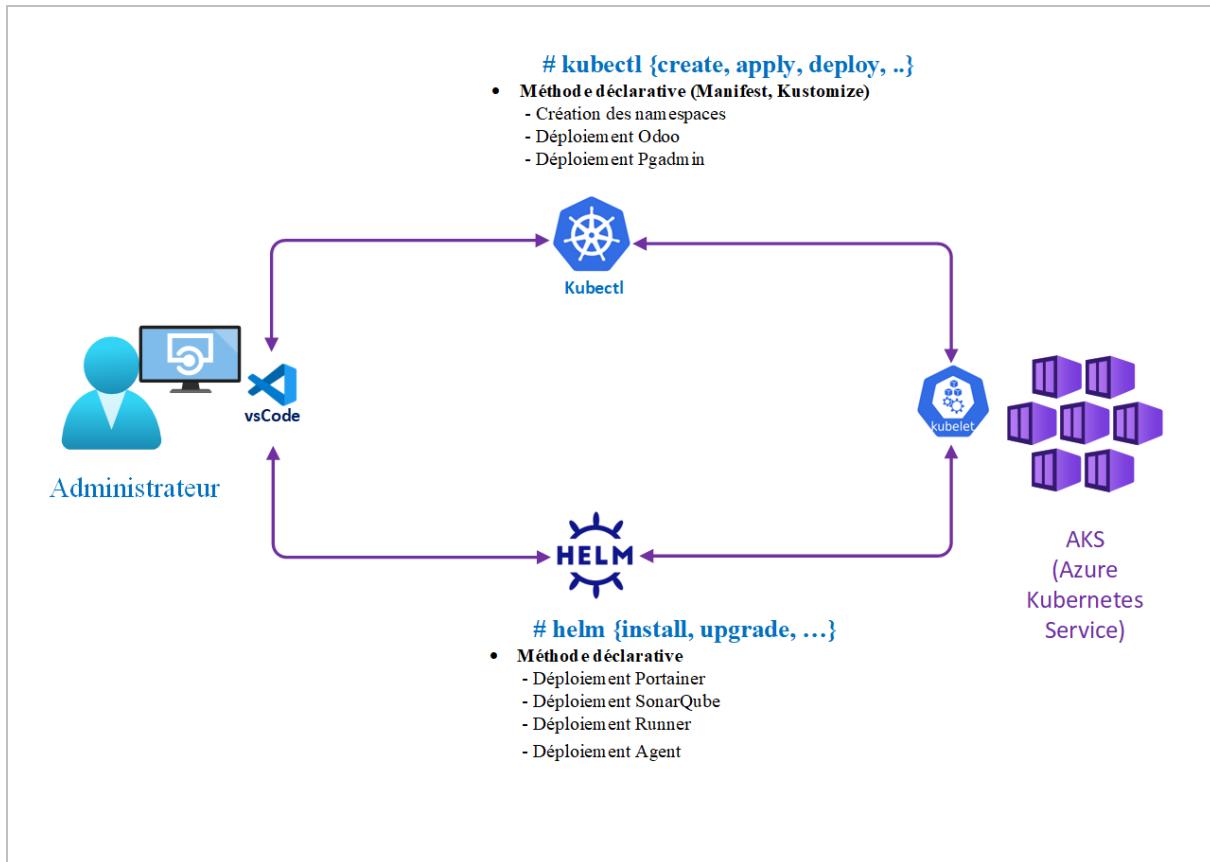


Figure 19 : Méthodes de déploiement utilisées

L'outil de ligne de commande kubectl prend en charge plusieurs façons différentes de créer et de gérer les objets Kubernetes.

Acteur :	Administrateur
Précondition :	Connexion au serveur Kubernetes.
Post condition :	Développement et préparation des configurations (manifestes, modèles, Chart Helm) requises pour le déploiement des composants nécessaires à la réalisation du projet.
Description :	L'administrateur doit préparer des configurations qui seront utilisées soit pour des déploiements manuels, soit pour des déploiements automatisés par pipeline CI/CD.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. L'administrateur crée les namespaces (dev, prod) dans lesquels les déploiements seront effectués.</li> <li>2. Le développeur crée un service Account qui sera utilisé par Gitlab CI pour exécuter les jobs de pipeline au niveau Kubernetes</li> <li>3. L'administrateur fait le déploiement avec le gestionnaire de package HELM :               <ol style="list-style-type: none"> <li>a. Portainer</li> <li>b. Ingress-Controller</li> </ol> </li> </ol>

	<ul style="list-style-type: none"> <li>c. Cert-manager</li> <li>d. PostgreSQL</li> <li>e. SonarQube</li> <li>f. Grafana &amp; Prometheus</li> </ul> <p>4. L'administrateur crée un modèle (Template) de déploiement avec l'outil Kustomize :</p> <ul style="list-style-type: none"> <li>a. Instance Odoo</li> <li>b. PGAdmin</li> </ul> <p>5. L'administrateur écrire le script du pipeline avec le respect d'ordre et dépendance des étapes.</p> <ul style="list-style-type: none"> <li>a. Intégrer kaniko pour faire le build</li> <li>b. Intégrer SonarQube</li> <li>c. Intégrer kubectl pour établir la connexion à nos clusters et faire la livraison et le déploiement.</li> </ul> <p>6. Ajouter notre cluster de développement à Portainer</p> <p>7. Fournir l'accès Portainer aux développeurs (Chef de projet)</p> <p>8. Fournir l'accès au Dashboard de monitoring aux développeurs (Chef de projet)</p>
Scénario Alternatif :	Côté Kubernetes, le ServiceAccount est incorrect ou mal configuré.
Scénario d'erreur :	L'administrateur commet une erreur d'accès et le job ne peut pas être exécuté.

Tableau 17 - Description textuelle du cas d'utilisation "Préparation des configurations de déploiement"

#### 3.4.4. Supervision

La plateforme de supervision est principalement utilisée pour surveiller l'état du cluster AKS, sa disponibilité et ses performances en termes de ressources (CPU, RAM, utilisation des disques, réseau), ainsi que l'état des Pods (activés, suspendus, en attente, désactivés) et leur utilisation des ressources CPU, RAM et réseau.

Le module de supervision est présenté dans le diagramme de cas d'utilisation de la figure 20.

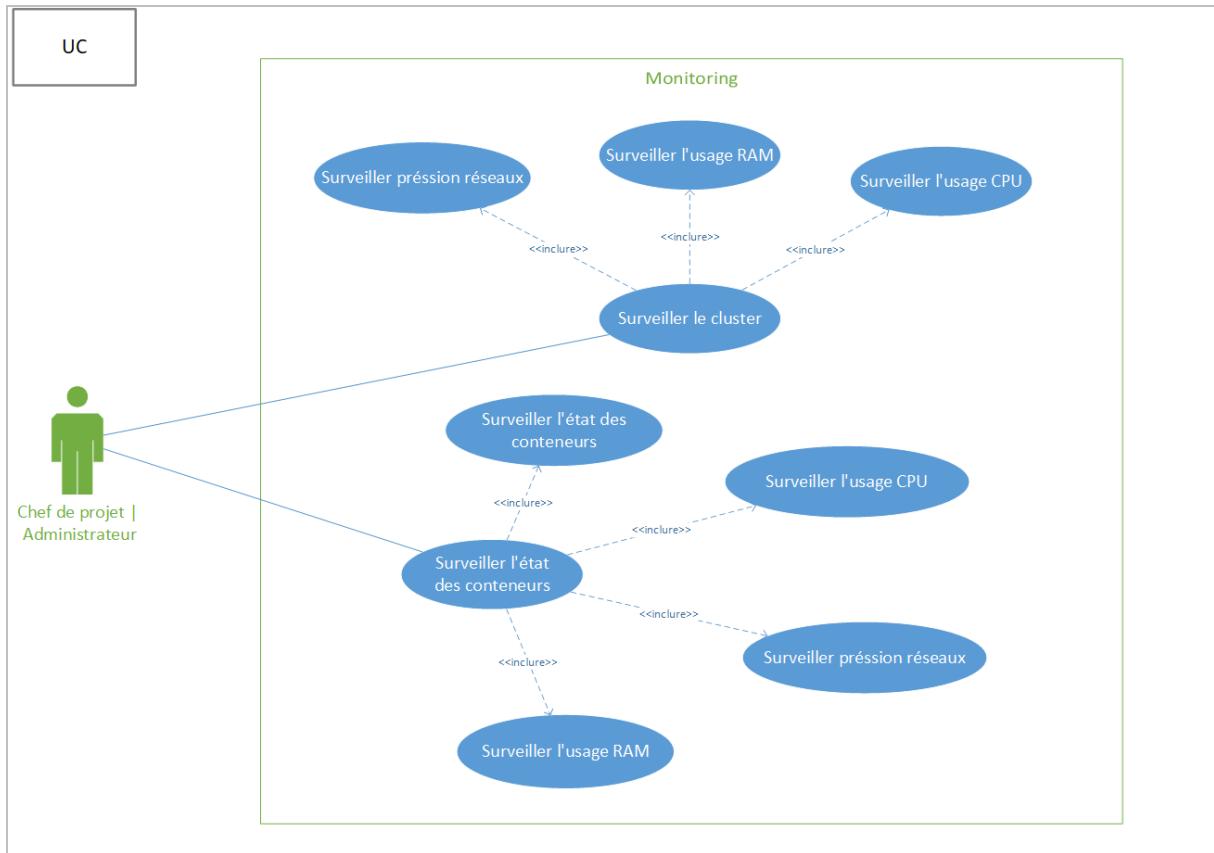


Figure 20 : Diagramme de cas d'utilisation "Supervision".

Acteur :	Administrateur
Précondition :	Connexion à l'interface de monitoring « Grafana & Prometheus » Connexion à l'interface Portainer.
Post condition :	Afficher une table de surveillance contenant les éléments décrit.
Description :	L'administrateur dispose de mesures sur l'usage du CPU, de la RAM et des E/S sur le réseau à côté des pods et du cluster AKS.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. L'administrateur sélectionne l'élément (Cluster, Namespace, Pod) qu'il souhaite visualiser.</li> <li>2. Le tableau de bord affiche des panneaux avec des graphiques décrivant l'état des ressources, l'utilisation du processeur, l'utilisation de la mémoire vive et l'occupation du réseau pour l'élément sélectionné.</li> </ol>
Scénario Alternatif :	<b>A1 : Email ou mot de passe non valide</b> 1. Message : Votre identifiant ou votre mot de passe est incorrect.
Scénario d'erreur :	L'administrateur commet une erreur d'accès.

Tableau 18 - Description textuelle du cas d'utilisation "Surveiller le cluster et les projets"

Acteur :	Développeur (Chef de projet)
Précondition :	<ul style="list-style-type: none"> <li>– Accès à l'interface de monitoring « Grafana &amp; Prometheus »</li> <li>– Accès à l'interface Portainer.</li> </ul>
Post condition :	Affiche un tableau de surveillance indiquant l'état du pod « actif », « arrêté », « en attente », « Inconnu », leurs usage des ressources CPU, RAM et réseau.
Description :	Les développeurs (chefs de projet) doivent avoir une visibilité globale grâce à des tableaux de bord de surveillance pour voir quels pods ne fonctionnent pas et quels pods fonctionnent.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Le chef de projet peut consulter l'historique des éléments sous monitoring.</li> <li>2. L'état des pods est affiché regroupé par projet au niveau du tableau de bord.</li> </ol>
Scénario Alternatif :	<b>A1 : Email ou mot de passe non valide</b> <ol style="list-style-type: none"> <li>1. un message d'erreur : Votre identifiant ou votre mot de passe est incorrect.</li> </ol>
Scénario d'erreur :	Le Chef de projet commet une erreur d'accès.

Tableau 19 - Description textuelle du cas d'utilisation "Surveiller l'état des Pods"

### 3.4.5. Pré-configuration

La partie pré-configuration consiste à mettre en place les plateformes nécessaires à la réalisation de notre projet.

Comme nous l'avons vu dans les objectifs et la contribution, la technologie de conteneurisation sera la base de nos déploiements pour toutes les applications du projet, pour cela nous avons choisi d'utiliser l'AKS (Azure Kubernetes Service). Pour gérer l'intégration et le déploiement continu, nous avons créé et configuré un compte sur la plateforme gitlab.com.

Nous illustrons la phase de pré configuration à l'aide de deux diagrammes de cas d'utilisation. La figure 21 montre les cas d'utilisation nécessaires à la mise en place d'un cluster AKS (Azure Kubernetes Service), tandis que la figure 23 montre les cas d'utilisation nécessaires à la création et à la configuration d'un compte sur la plateforme gitlab.com.

### Pré-Configuration: Création et configuration d'un cluster AKS

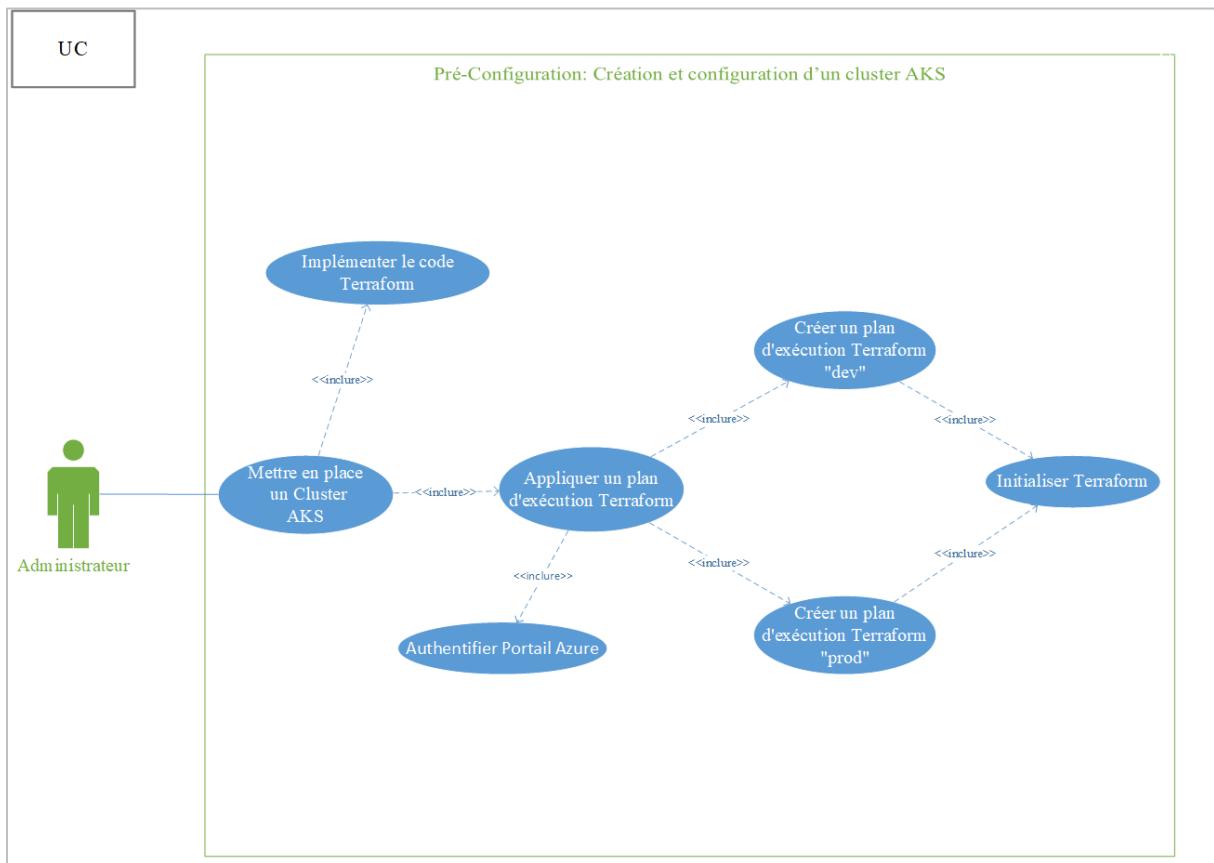


Figure 21 : Diagramme de cas d'utilisation "Création et configuration d'un cluster AKS"

Acteur :	Administrateur
Précondition :	<ul style="list-style-type: none"> <li>Authentifier au portail Azure Cloud,</li> <li>Terraform installé et configuré,</li> <li>Kubernetes command-line tool (<code>kubectl</code>)</li> </ul>
Post condition :	Automatisation et approvisionnement du service AKS (Azure Kubernetes Service) avec l'outil d'orchestration Terraform.
Description :	<p>L'administrateur doit développer leur plan Terraform qui décrire les modules (clusters) et leurs groupe de ressource, le nombre des nœuds et les variables pour chacun d'eux.</p> <ol style="list-style-type: none"> <li>Module : <ul style="list-style-type: none"> <li>Cluster_name : dev_env</li> <li>Node_count : 1 ou plus</li> <li>Vm_size : standard_d2_v2   standard_d11_v2</li> </ul> </li> </ol>
Scénario nominal :	<ol style="list-style-type: none"> <li>Ouvrir l'IDE vsCode, où nous gère et développe notre code Terraform.</li> <li>Authentification au portail azure.</li> <li>Développement des modules</li> </ol>

	<ol style="list-style-type: none"> <li>4. Créer un plan d'exécution avec la commande « terraform plan », ce dernier nous aider à pré visualiser les modifications que Terraform prévoit d'apporter à votre infrastructure.</li> <li>5. Si le résultat est convient à nos besoin, nous appliquons le plan avec la commande « terraform apply »</li> <li>6. En ajoute le chemin du fichier « kubeconfig » généré dans le variable d'environnement « KUBECONFIG » de notre système.</li> <li>7. Vérifier qu'un nouveau contexte a été ajouté « <i>kubectl config get-context</i> »</li> <li>8. Basculer vers le nouveau contexte « <i>kubectl config set-context ciamcluster-dev</i> »</li> <li>9. Vérifier le nouveau contexte avec la « <i>kubectl config current-context</i> ».</li> </ol> <p>Nous pouvons maintenant gérer notre cluster à partir de la ligne de commande.</p>
Scénario Alternatif :	<p><b>A1 : L'authentification n'a pas abouti</b></p> <ol style="list-style-type: none"> <li>1. Message : le login ou le mot de passe est incorrect.</li> </ol> <p><b>A2 : Erreur de création de plan</b></p> <ol style="list-style-type: none"> <li>1. Message : les fichiers nécessaires ne sont pas trouvés</li> </ol> <p><b>A3 : Problème d'accès au cluster AKS</b></p> <ol style="list-style-type: none"> <li>1. Message : Commande "Kubectl" inconnue</li> </ol>
Scénario d'erreur :	<ul style="list-style-type: none"> <li>- Les informations d'accès ne sont pas correctes</li> <li>- Terraform n'a pas réussi à télécharger les paquets du fournisseur</li> <li>- Kubectl n'est pas installé et configuré sur la machine</li> </ul>

Tableau 20 - Description textuelle du cas d'utilisation "Mise en place un cluster AKS"

La figure 22 illustre les étapes de l'exécution du code Terraform pour créer et provisionner le cluster AKS (Azure Kubernetes Service).

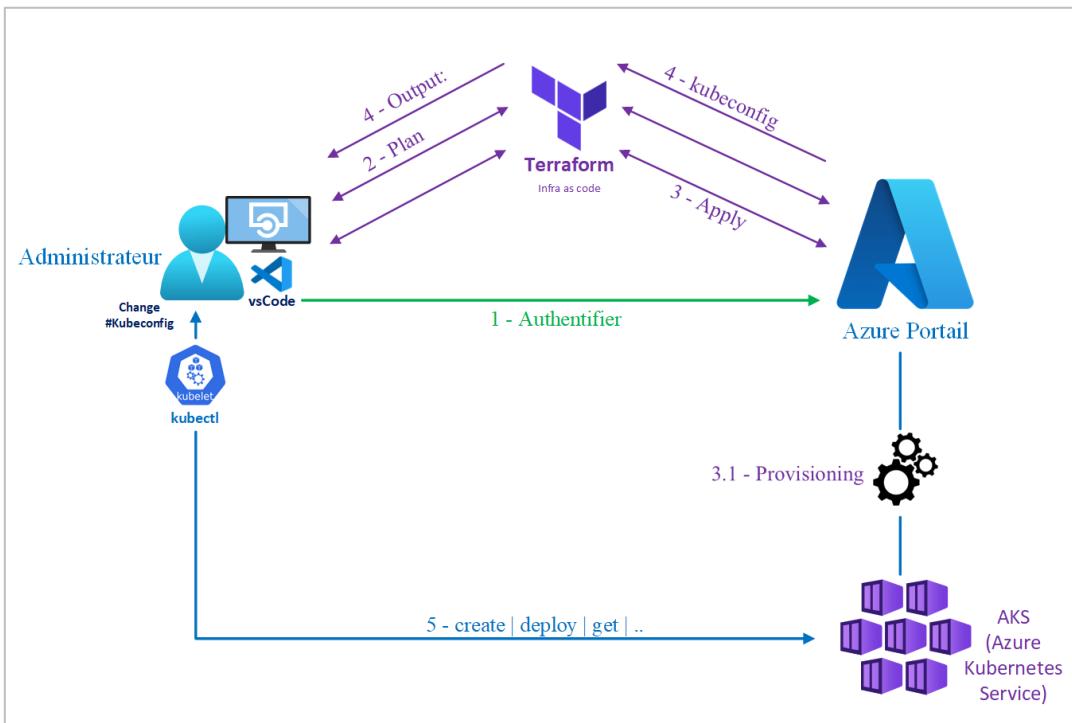


Figure 22 : Architecture de déploiement et d'approvisionnement d'AKS avec Terraform

#### Pré-Configuration: Création et configuration d'un compte gitlab.com

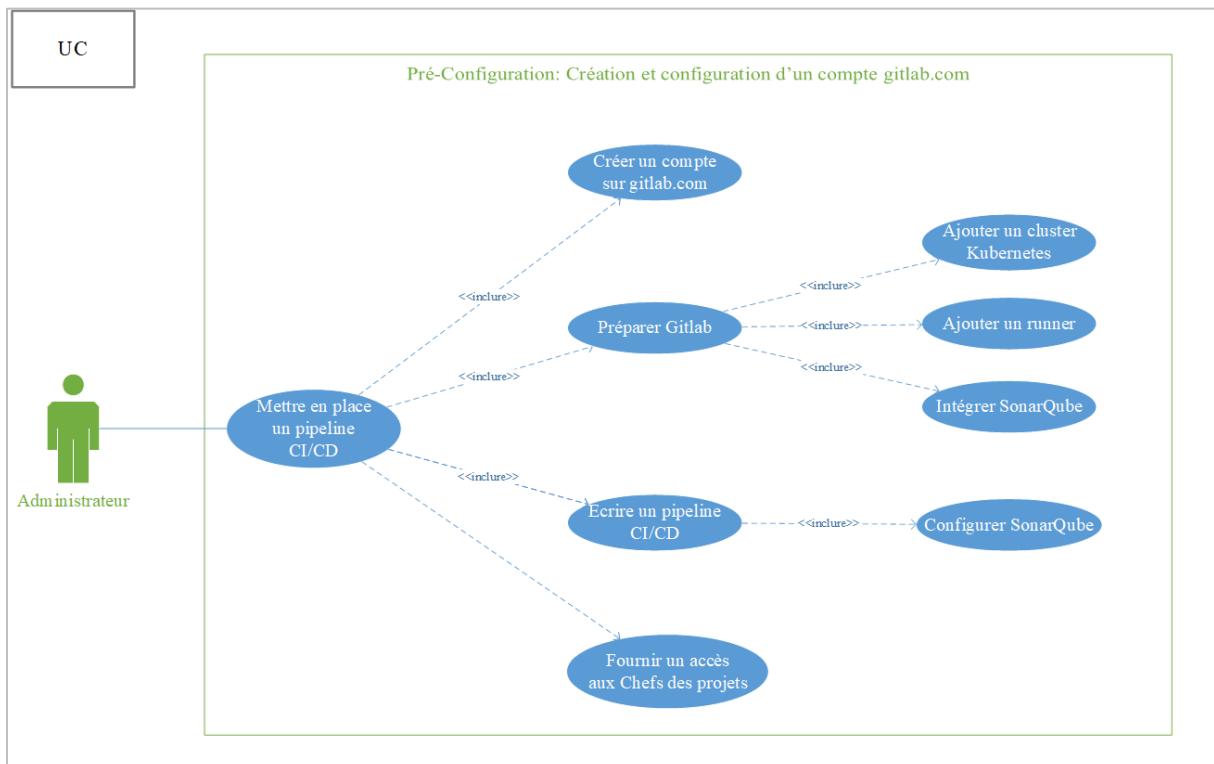


Figure 23 : Diagramme de cas d'utilisation "Création et configuration d'un compte Gitlab"

<b>Acteur :</b>	<b>Administrateur</b>
Précondition :	Accès au site web gitlab.com
Post condition :	Création d'un compte sur gitlab.com.
Description :	L'administrateur doit avoir un compte sur la plateforme managé Gitlab.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Accéder à <a href="#">gitlab.com</a></li> <li>2. Cliquer sur « Inscription »</li> <li>3. Remplir le formulaire et valider</li> <li>4. Confirmer l'inscription via le lien existe dans l'email reçu.</li> <li>5. Accéder au compte gitlab.com.</li> </ol>
Scénario Alternatif :	<b>A1 : Compte non activé</b> <ol style="list-style-type: none"> <li>1. Message : Activez votre compte en utilisant le lien envoyé par e-mail.</li> </ol>
Scénario d'erreur :	<ul style="list-style-type: none"> <li>- Boîte aux lettres invalide</li> </ul>

Tableau 21 - Description textuelle du cas d'utilisation "Création et configuration d'un compte gitlab.com"

<b>Acteur :</b>	<b>Administrateur</b>
Précondition :	Accès au compte gitlab.com, un cluster Kubernetes est prêt.
Post condition :	Un cluster Kubernetes sera connecté à la plateforme gitlab.com.
Description :	Avec l'ajout du cluster Kubernetes à la plateforme gitlab.com, ce dernier sera disponible pour l'exécution des jobs de déploiement.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Connexion à la plateforme gitlab.com</li> <li>2. Aller vers un projet, puis « Opérations » → « Clusters Kubernetes »</li> <li>3. Cliquer sur « Connecter un cluster », saisissez le nom de l'agent, puis cliquer sur « Créer l'agent ».</li> <li>4. Valider la création par l'enregistrement.</li> <li>5. Une fenêtre contextuelle s'affiche, contenant un « Token » et les commandes nécessaires au déploiement d'un agent dans le cluster Kubernetes à l'aide du gestionnaire de paquets « Helm ».</li> <li>6. Ouvrez un terminal et connectez-vous au cluster Kubernetes.</li> <li>7. Exécutez les commandes fournies par l'étape 5.</li> <li>8. Vérifier que votre agent est connecté.</li> </ol>
Scénario Alternatif :	<b>A1 : La connexion n'a pas abouti</b> <ol style="list-style-type: none"> <li>1. Message : Temps de réponse écoulé</li> </ol>
Scénario d'erreur :	<ul style="list-style-type: none"> <li>- Pas d'accès disponible vers le cluster AKS</li> </ul>

Tableau 22 - Description textuelle du cas d'utilisation "Ajouter un cluster Kubernetes à gitlab.com"

<b>Acteur :</b>	<b>Administrateur</b>
Précondition :	Accès au compte gitlab.com, un « Runner » est prêt.
Post condition :	Ajouter un Runner à la plateforme gitlab.com.
Description :	Avec l'ajout d'un Runner à la plateforme gitlab.com, ce dernier sera disponible pour l'exécution des jobs de pipeline.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Connexion à la plateforme gitlab.com</li> <li>2. Au niveau de votre projet « Paramètres » → CI/CD → Runner</li> <li>3. Cliquer sur « Nouvel runner du projet »</li> <li>4. Sélectionner le système d'exploitation « linux », « Kubernetes » pour le conteneur.</li> <li>5. Définir une étiquette « Kube-runner » par exemple.</li> <li>6. Enregistrer</li> <li>7. Suite à l'enregistrement, une page s'affiche comporte un « Token » et les commandes nécessaires afin de d'ajouter le « Runner » déjà déployé.</li> <li>8. Ouvrir une console sur le « Runner », puis exécutez les commandes (point 7).</li> <li>9. Vérifier que le « Runner » est connecté.</li> </ol>
Scénario Alternatif :	<b>A1 : La connexion n'a pas abouti</b> <ol style="list-style-type: none"> <li>1. Message : Temps de réponse écoulé</li> </ol>
Scénario d'erreur :	<ul style="list-style-type: none"> <li>- Pas d'accès disponible vers le Runner.</li> </ul>

Tableau 23 - Description textuelle du cas d'utilisation "Ajouter un Runner à gitlab.com"

<b>Acteur :</b>	<b>Administrateur</b>
Précondition :	Accès au compte gitlab.com, une instance SonarQube est prête.
Post condition :	Ajouter l'instance SonarQube à la plateforme gitlab.com.
Description :	Avec l'intégration de SonarQube avec la plateforme gitlab.com, ce dernier pourra voir nos projets dans le référentiel Git, et par la suite, nous pourrons alors configurer l'analyse pour chaque projet.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Connexion à l'interface « SonarQube »</li> <li>2. Cliquer sur « Administration », puis « DevOps Platform Integrations ».</li> <li>3. Sélectionner « Gitlab », puis « Create configuration »</li> <li>4. Remplir le formulaire :             <ol style="list-style-type: none"> <li>a. Nom de la configuration : MonProjet</li> <li>b. GitLab API URL : <a href="https://gitlab.com/api/v4">https://gitlab.com/api/v4</a></li> <li>c. Personal Access Token : Token (voir les étapes ci-dessous)</li> </ol> </li> </ol>

	<p>Création d'un Personal Access Token :</p> <ol style="list-style-type: none"> <li>1. Connexion à gitlab.com</li> <li>2. Aller vers « Préférences » → « Jetons d'accès »</li> <li>3. Cliquer sur « Ajouter un nouveau jeton »             <ol style="list-style-type: none"> <li>a. Nom : SonarQube</li> <li>b. Date d'expiration : 2024-01-01</li> <li>c. Sélectionner les portées : {read_api}</li> <li>d. Rôle : Reporter</li> <li>e. Valider</li> <li>f. Copier le jeton (Token) généré.</li> </ol> </li> <li>d. Valider &amp; Vérifier configuration</li> </ol>
Scénario Alternatif :	<p><b>A1 : Connexion non disponible</b></p> <ol style="list-style-type: none"> <li>1. Message : Destination inconnue</li> </ol>
Scénario d'erreur :	<ul style="list-style-type: none"> <li>- Les informations d'accès (Token) sont incorrectes</li> <li>- Le jeton d'accès (Token) a expiré</li> </ul>

Tableau 24 - Description textuelle du cas d'utilisation "Intégrer SonarQube à gitlab.com"

Acteur :	Administrateur
Précondition :	SonarQube est intégré à gitlab.com et peut voir les projets concernés par l'analyse statique du code.
Post condition :	Ajouter la configuration de l'analyse à votre pipeline CI/CD déjà défini dans le fichier « .gitlab-ci.yml ».
Description :	Une fois SonarQube intégré à gitlab.com, nous pouvons configurer l'analyse pour chaque projet. Cette configuration se fera dans l'étape « scan » de notre Pipeline.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Connexion à l'interface « SonarQube »</li> <li>2. Cliquer sur « Projets », puis « Import from Gitlab ».</li> <li>3. Cliquer sur « Import » pour le projet concerné</li> <li>4. Sélectionner « Utiliser le paramètre global », puis cliquer sur « Créer Projet ».</li> <li>5. La méthode de l'analyse avec « Gitlab CI »</li> <li>6. Appliquer les étapes affichées dans cette page :             <ol style="list-style-type: none"> <li>a. Ajouter des variables d'environnement Dans GitLab, allez dans Paramètres → CI/CD → Variables</li> <li>b. Créer ou mettre à jour les fichiers de configuration                     <ol style="list-style-type: none"> <li>i. sonar-project.properties</li> <li>ii. .gitlab-ci.yml</li> </ol> </li> </ol> </li> </ol>

Scénario Alternatif :	<b>A1 : L'importation de projets n'est pas disponible</b> 1. Message : vous n'avez pas des droits d'accès
Scénario d'erreur :	– Les informations d'accès dans le fichier "sonar-project.properties" ont été modifiées.

Tableau 25 - Description textuelle du cas d'utilisation "Configurer l'analyse pour un projet"

Acteur :	<b>Administrateur</b>
Précondition :	Accès au compte gitlab.com
Post condition :	Fournir un accès aux chefs des projets.
Description :	Permettre aux chefs de projet de créer leurs projets, d'inviter l'équipe de développement appropriée et d'assurer une bonne collaboration entre tous les membres.
Scénario nominal :	<ol style="list-style-type: none"> <li>1. Connexion à la plateforme gitlab.com</li> <li>2. Cliquer sur la signe « + » en haut à gauche, « nouvelle groupe ».</li> <li>3. Remplir les informations nécessaires :             <ol style="list-style-type: none"> <li>a. Nom du groupe : DevTeam</li> <li>b. Niveau de visibilité : Privé</li> <li>c. Rôle: Devops Engineer</li> <li>d. Servira à : Une raison différente</li> <li>e. Membres : liste des boîtes mails des utilisateurs (Chefs des projets) a invité.</li> </ol> </li> <li>4. Un email sera envoyé aux chefs des projets</li> </ol>
Scénario Alternatif :	<b>A1 : Échec de la connexion</b> 1. Message : vérifiez votre connexion internet
Scénario d'erreur :	– Problème de connexion internet.

Tableau 26 - Description textuelle du cas d'utilisation "Fournir l'accès aux chefs des projets"

# Chapitre 4

## Architecture et conception détaillée

### 4.1. Introduction

Dans ce qui suit, nous nous intéressons à l'architecture globale du système, puis nous décrivons la vue dynamique de chaque volet du système, où nous découvrons les processus des sous-systèmes, les relations entre eux et leurs conditions d'exécution.

### 4.2. Architecture générale de la solution

Nous avons choisi un environnement répondant aux exigences techniques et fonctionnelles de la solution, basé sur différents outils et modèles cloud SaaS et PaaS, en appliquant l'approche DevOps pour assurer une meilleure collaboration entre les équipes, des économies de temps et de coûts, de l'innovation, une détection rapide des problèmes et plus encore pour l'entreprise hôte.

Le diagramme d'architecture de la figure 24 met en évidence toutes les plateformes et tous les services utilisés pour mettre en œuvre la solution, ainsi qu'une illustration du flux de travail du processus de la chaîne CI/CD.

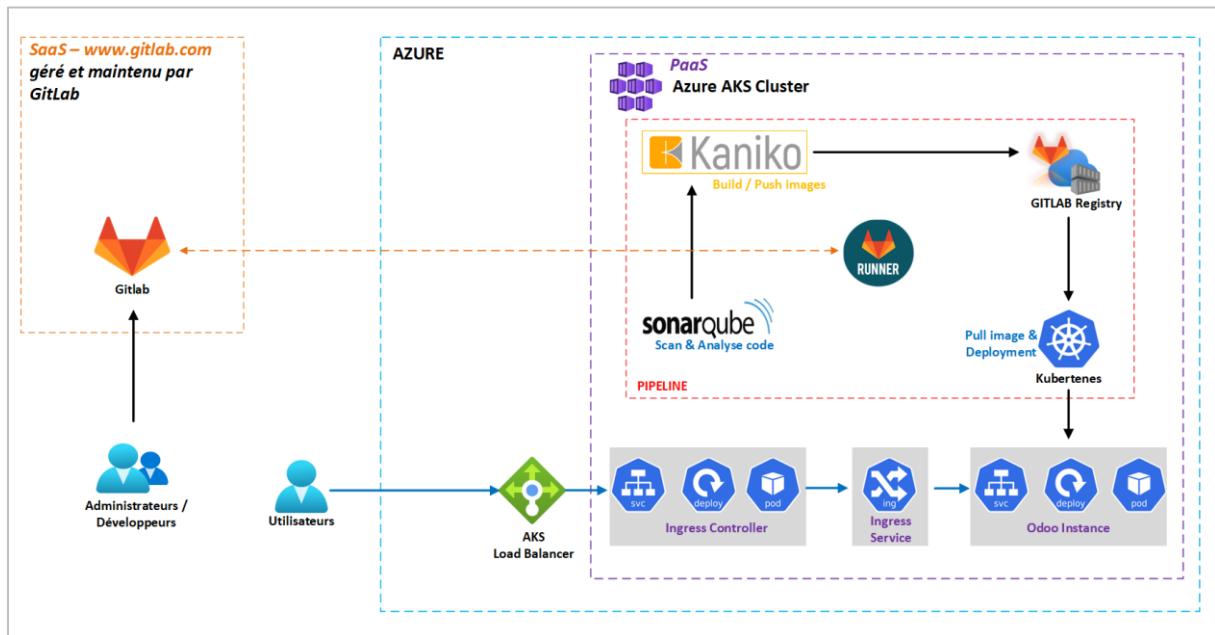


Figure 24 : Diagramme d'architecture générale

La figure 25 montre le diagramme de déploiement qui illustre notre architecture et les différents protocoles utilisés pour la communication des entités.

Nous notons que les services Gitlab Git, Gitlab-CI et Gitlab registry sont fournis de manière centralisée par GitLab Platform, qui est un service managé.

Le protocole HTTP est utilisé pour la communication entre les différents Pods.

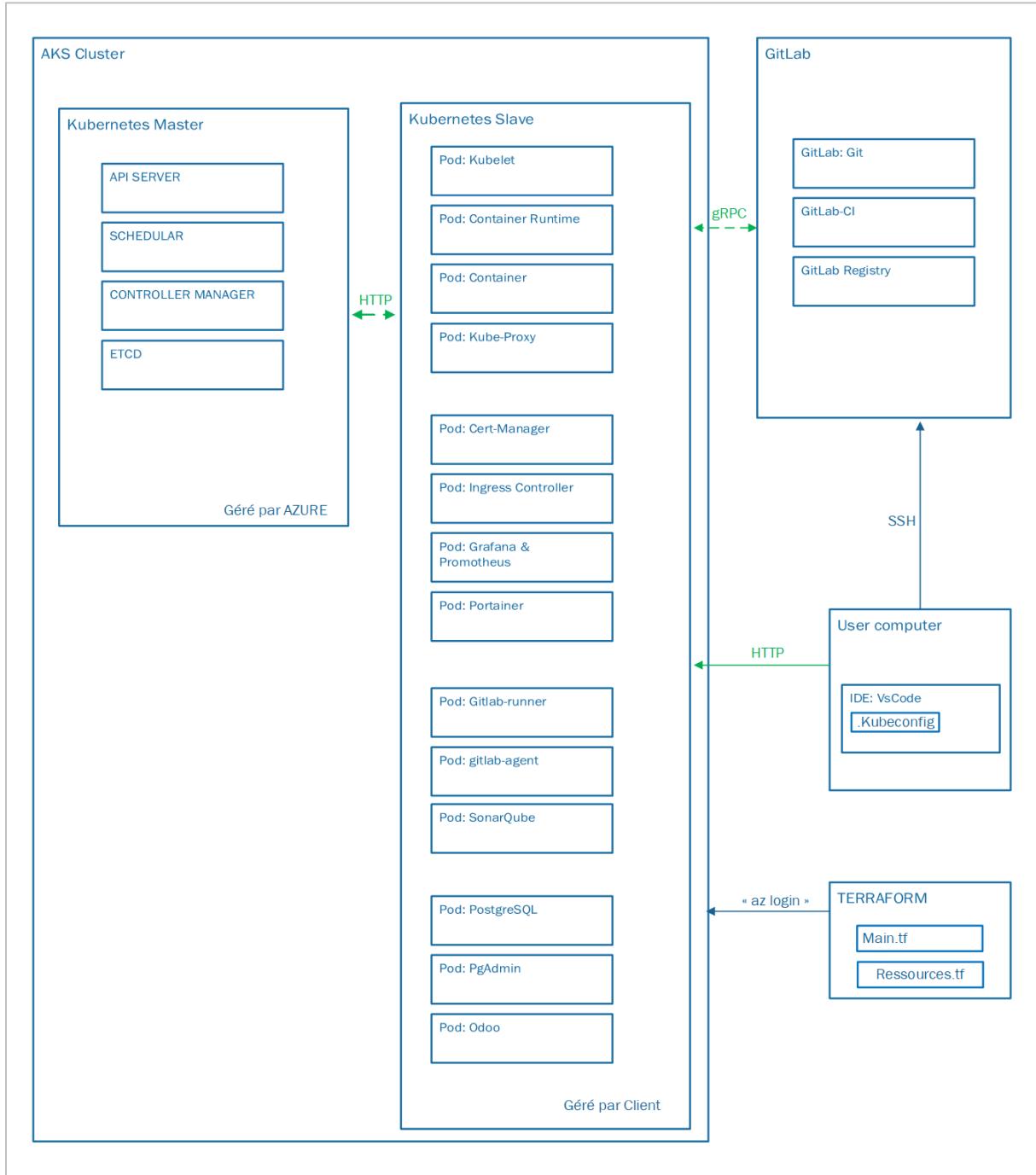


Figure 25 : Diagramme de déploiement et d'approvisionnement pour l'architecture à base de Kubernetes

Le déploiement et le provisionnement d'un cluster AKS dans l'environnement cloud Azure sont réalisés à l'aide de l'outil Terraform, qui utilise l'approche « Infrastructure as Code ».

Les déploiements effectués au niveau du cluster AKS concernent la gestion du cluster (sécurité d'accès aux espaces du nom, supervision), les processus de communication entre les plateformes (gitlab-agent, gitlab-runner), les outils et les travaux requis dans le pipeline (Sonarqube, kaniko, kubectl) et enfin les services requis pour l'exécution des instances Odoo et l'exploration de la base de données (PostgreSQL, PgAdmin).

### 4.3. Aspect dynamique de la solution

Nous présentons pour chaque volet du système une vue dynamique afin d'expliquer la technique et l'approche appliquées pour préparer l'environnement et exposer les interactions des processus utilisés dans la chaîne d'outils DevOps.

#### 4.3.1. Vue dynamique du volet "Intégration continue"

Le diagramme d'activités présenté dans la figure 26 nous explique la logique opérationnelle des différents processus impliqués dans le pipeline d'intégration continue.

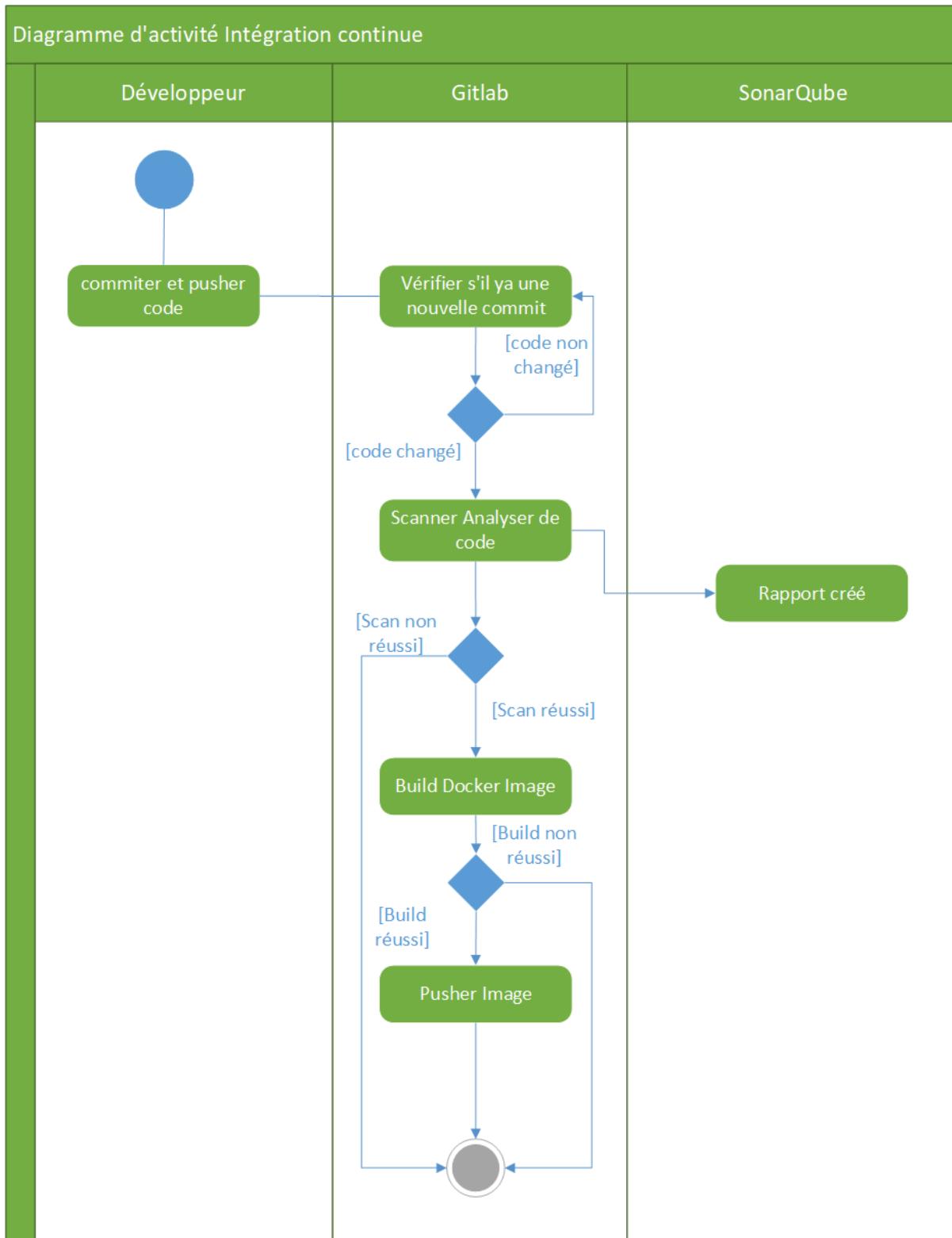


Figure 26 : Processus d'Intégration continue

#### 4.3.2. Vue dynamique du volet "Déploiement continu"

La figure 27 illustre le diagramme d'activité, qui explique la vue dynamique du processus de déploiement continu, ci-dessous plus d'informations sur les conditions et les acteurs impliqués dans ce processus.

Comme condition préalable, le build déclenchée précédemment doit s'exécuter avec succès, ainsi, le nœud master Kubernetes sélectionne le nœud slave pour créer le déploiement. La sélection dépend de la configuration des ressources définie pour la création du Pod et sur des ressources disponibles sur les nœuds du cluster. Le Pod se voit attribuer une adresse IP privée, qui n'est connue que du cluster Kubernetes, une fois qu'un nœud a été sélectionné. Kubernetes crée ensuite un conteneur en instanciant l'image Docker précédemment récupérée du registre gitlab après une build réussie.

Enfin, pour que les utilisateurs puissent accéder à leurs instances d'application, une route doit être créée pour chaque service exposé par kubernetes.

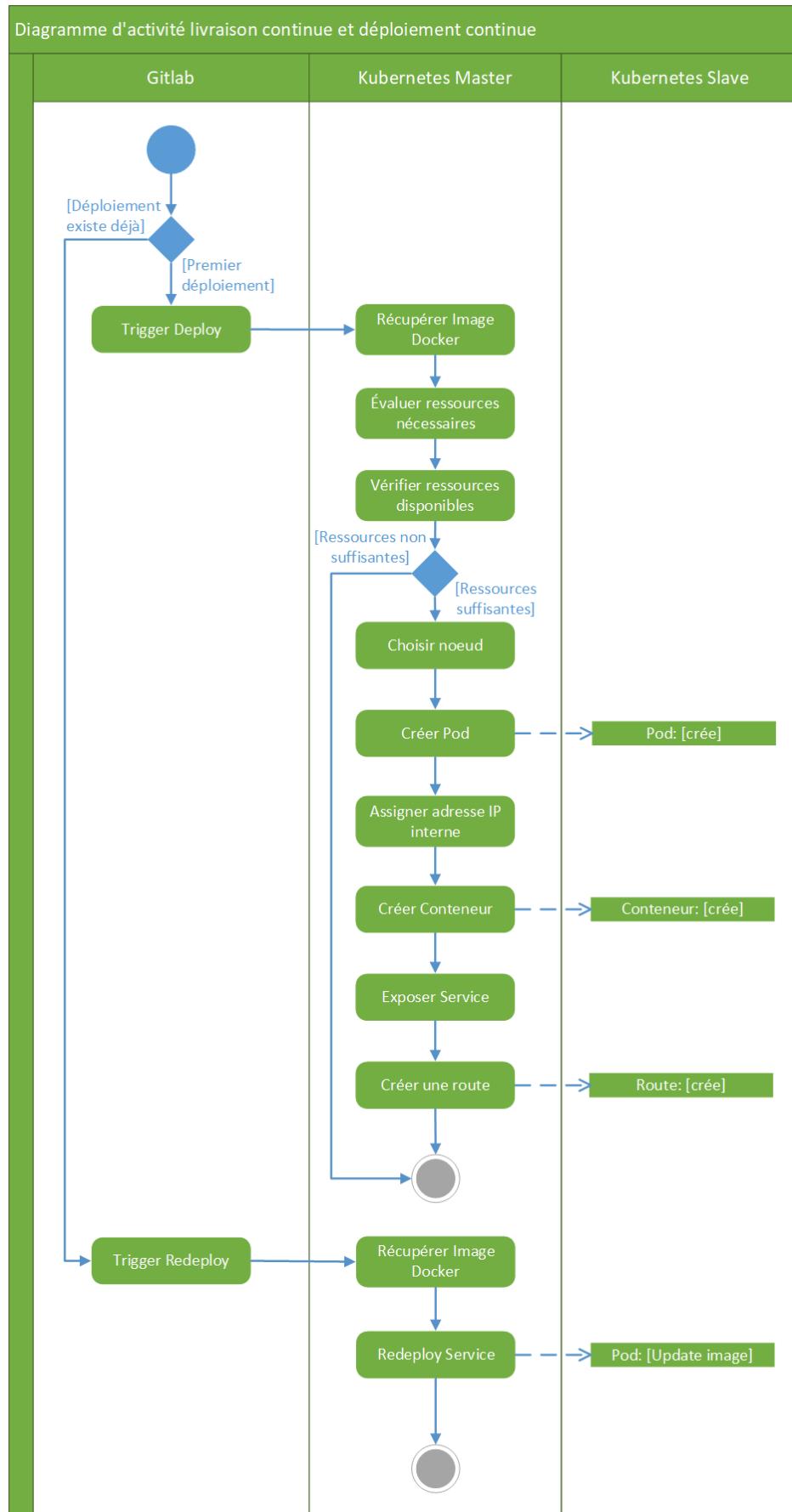


Figure 27 : Processus de déploiement d'une application

#### 4.3.3. Vue dynamique du volet "Configuration"

Nous définissons ci-dessous une liste d'éléments qui présente les objets nécessaires et les concepts techniques utilisés dans notre solution.

- Dockerfile : Ce fichier contient toutes les commandes dont nous avons besoin pour assembler une image Docker techniquelement conforme.
- Namespace : est un mécanisme qui permet d'isoler des groupes de ressources au sein d'un même cluster, en fournissant un périmètre pour les pods, les services et les déploiements dans kubernetes.
- ServiceAccount : présente le compte d'accès utilisé par la plateforme Gitlab pour se connecter à l'environnement kubernetes (développement ou production) afin d'exécuter les travaux du pipeline.
- ClusterRole : est la définition des droits d'accès (Namespace, Deployments, PersistantVolume, Ingress) attribués au ServiceAccount.
- Docker Registry : est l'objet qui nous permet de définir les informations d'accès pour l'utilisation d'une image docker. Il sera également assigné à notre ServiceAccount avec la spécification « imagePullSecrets ».
- Template de Déploiement : est la définition des pods, des services, du routage, des ressources et de l'état souhaité dont nous avons besoin pour faire fonctionner notre instance Odoo et PgAdmin.
- Script Pipeline « gitlab-ci.yaml » : Ce fichier contient un script définissant la structure et l'ordre des travaux qui doivent être exécutés pour chaque commit.
- Chart Helm : La définition de nos charts (fichier .yaml) nous permet de spécifier la version et les options du paquet à installer avec Helm, le gestionnaire de paquets pour Kubernetes.

Nous illustrons les cas d'usage présents dans le module Configuration à travers le diagramme d'activité de la Figure 28.

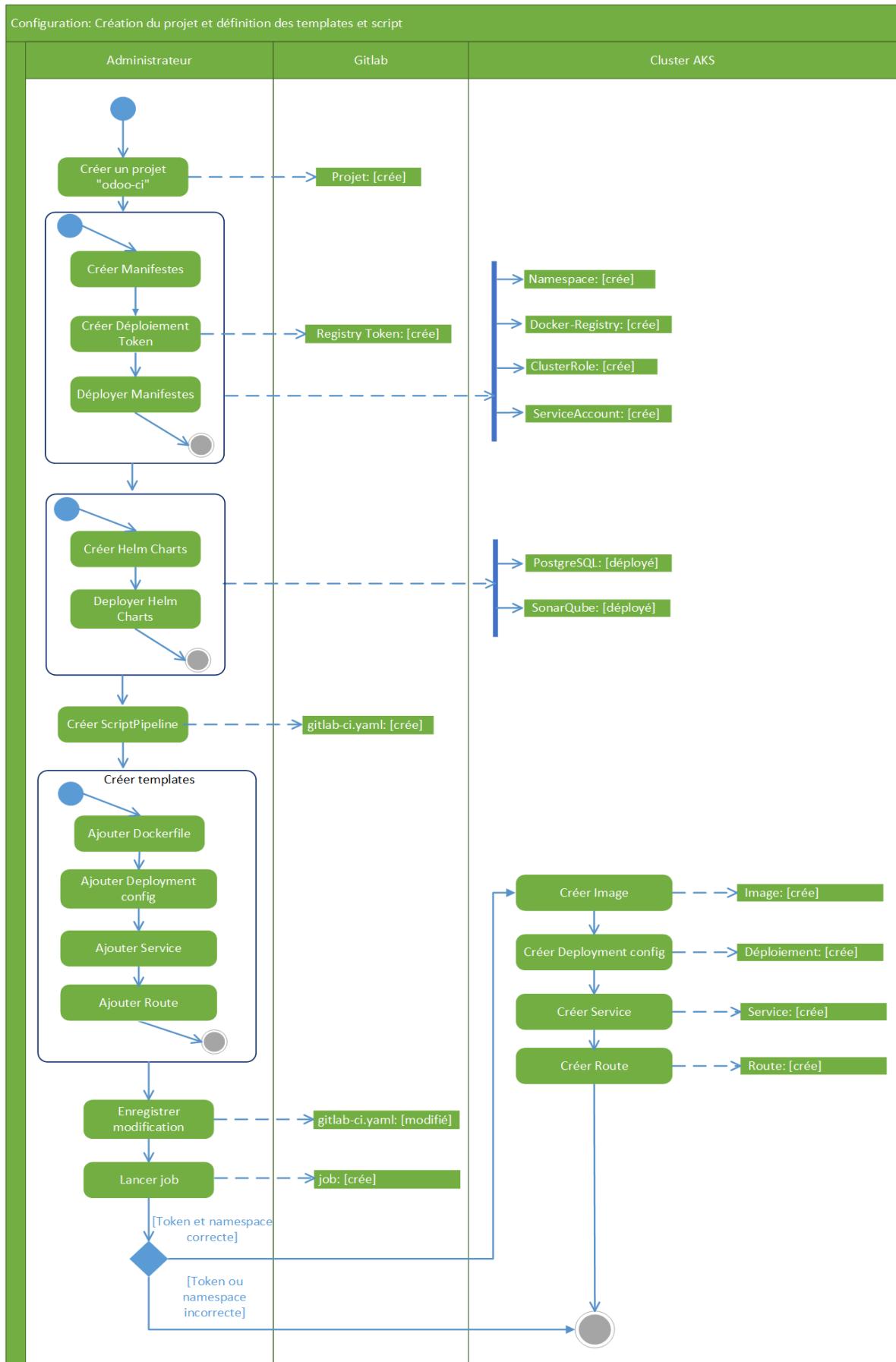


Figure 28 : Processus de création et définition d'un projet

#### 4.3.4. Vue dynamique du volet "Pré-configuration"

Nous avons également utilisé les diagrammes d'activités pour présenter ce module. Les figures 29, 30 et 32 expliquent les étapes de mise en place et de provisionnement d'un cluster AKS dans le cloud Azure avec l'outil Terraform, le déploiement des solutions de sécurité, de supervision et d'accès dans ce dernier. La création d'un compte Gitlab, puis la configuration et l'intégration avec des outils sert à l'exécution de notre pipeline.

Le volet de pré configuration comprend également des configurations pour extraire des métriques, comme le montre la figure 31.

##### 4.3.4.1. *Mise en place et approvisionnement d'un cluster AKS*

La figure 29 illustre les trois principales étapes de la création et du provisionnement d'un cluster AKS dans le cloud Azure, ces étapes sont :

- Authentification au portail Azure
- Après initialisation de Terraform, la création, révision ou mise à jour d'un plan.
- L'approvisionnement (création, modification, suppression) d'un cluster.

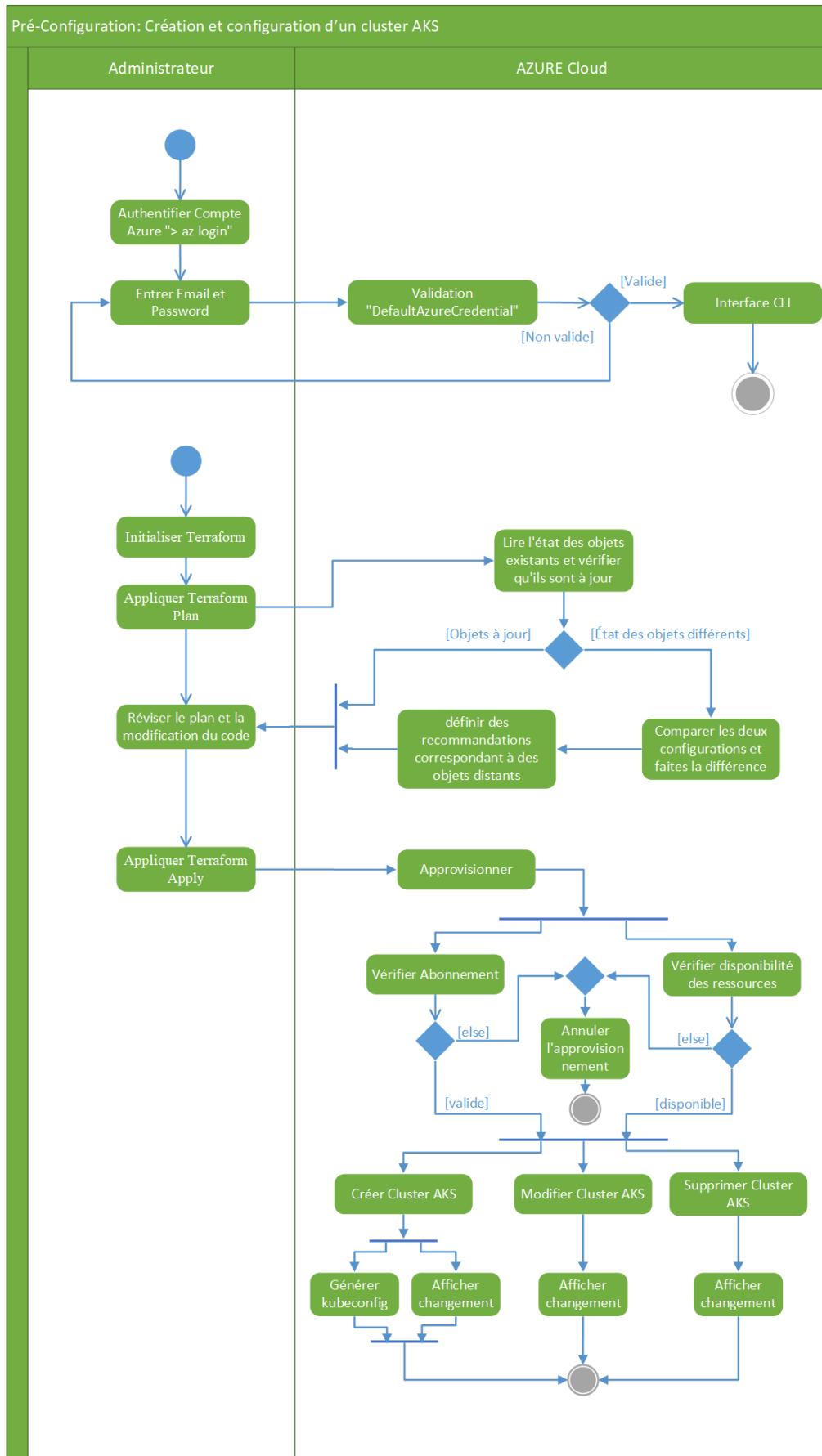


Figure 29 : Processus de mise en place et approvisionnement d'un Cluster AKS

L'administration, la supervision et la sécurité sont des composantes essentielles de notre AKS, et la figure 30 illustre le déploiement d'outils pour répondre à ces besoins. Ces outils sont :

- Portainer
- Grafana et Prometheus
- Cert-Manager
- Ingress-Controller

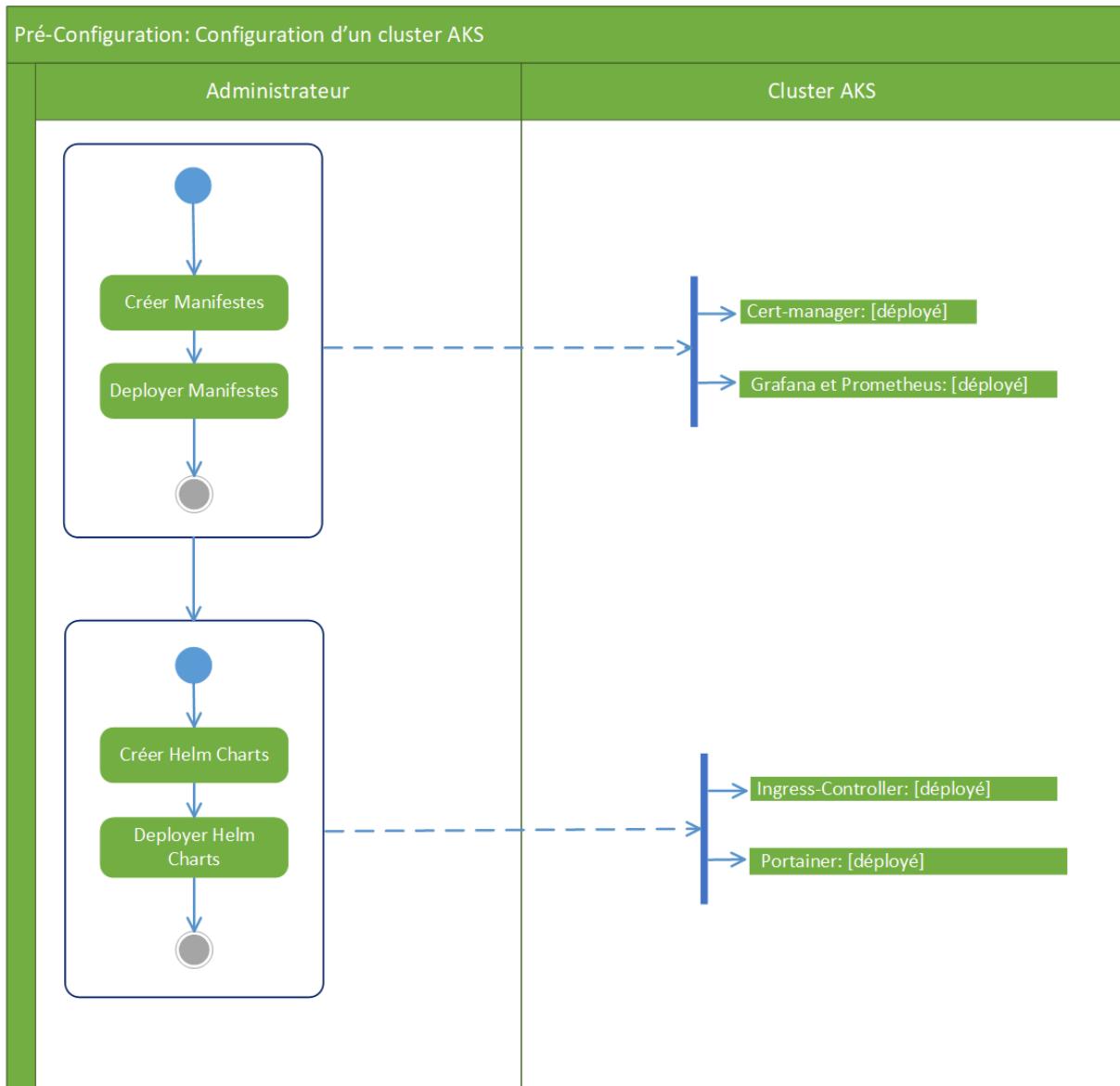


Figure 30 : Configuration de la sécurité, de la supervision et de l'accès

#### 4.3.4.2. Configuration des métriques

La surveillance des ressources nous permet de mesurer les performances globales de notre cluster AKS en affichant des métriques sur l'utilisation actuelle des processeurs, de la RAM et même des disques sous différentes perspectives (nœuds, pods ou même l'ensemble du

cluster de serveurs). La solution que nous avons implémentée sur Kubernetes pour effectuer cette surveillance est constituée des piles technologiques Prometheus et Grafana. Le schéma ci-dessous illustre le fonctionnement de ces outils :

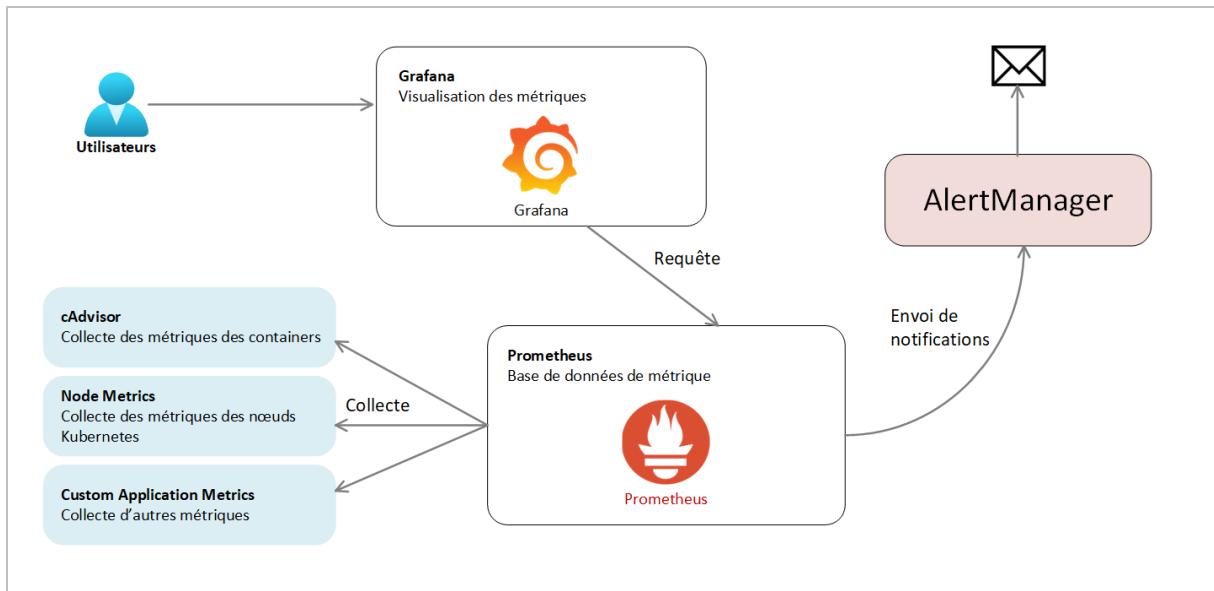


Figure 31 : Architecture de la solution de supervision des ressources

Prometheus est le composant central de cette solution et permet le stockage et l'analyse de données en série chronologique. Pour collecter ces données, il fait appel à plusieurs types de composants. L'outil « cAdvisor » permet de collecter les métriques provenant des conteneurs tandis que le « Node Exporter » permet de collecter les métriques au niveau du système d'exploitation (Nœud). Pour surveiller ces métriques, un « AlertManager » est utilisé et permet d'envoyer un rapport par courriel ou par messagerie instantanée. Finalement, l'outil Grafana effectue des requêtes à Prometheus et permet de visualiser les données sous forme de graphes, tel qu'illustré dans la capture d'écran suivante :

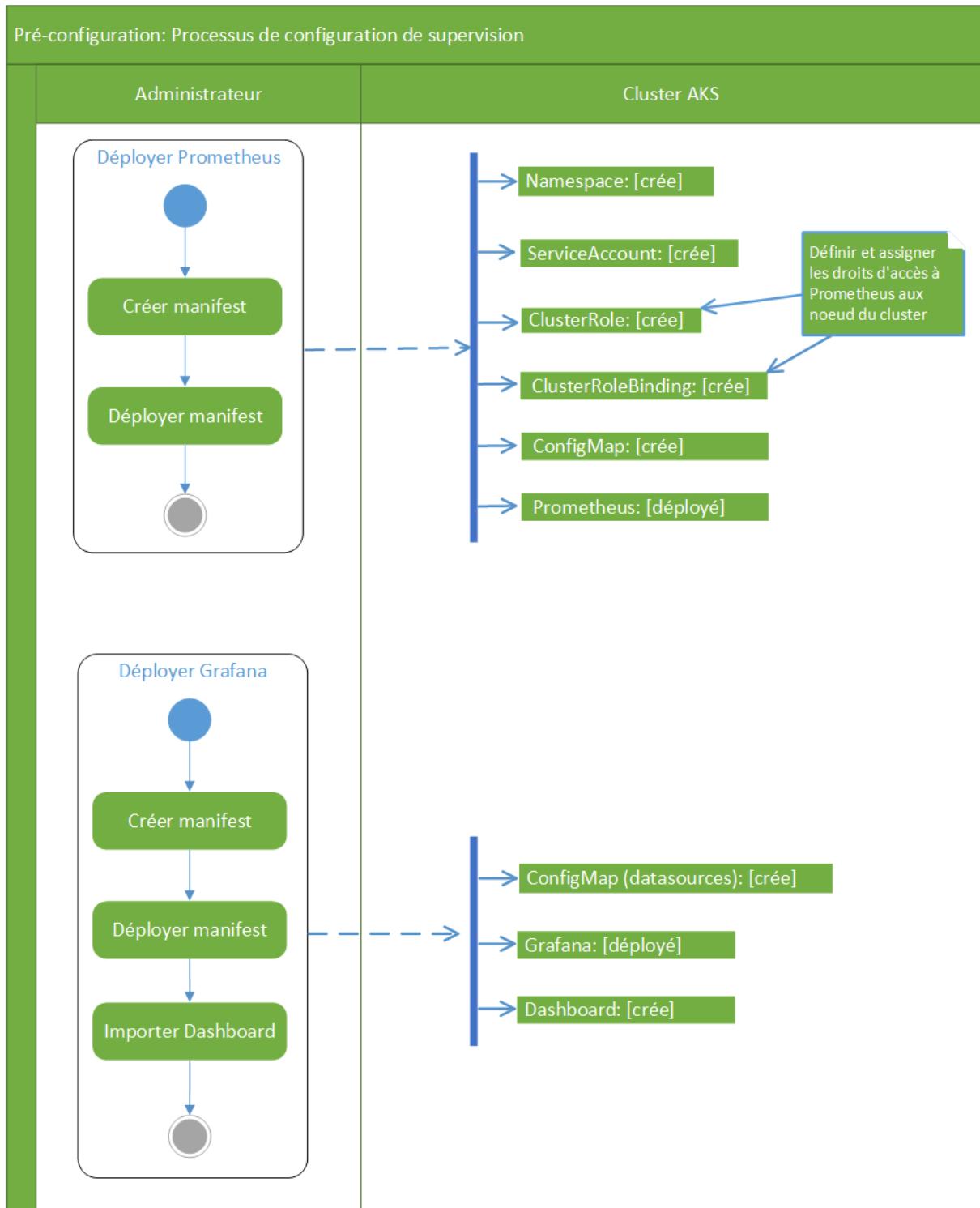


Figure 32 : Processus de configuration de supervision

#### 4.3.4.3. Création et configuration d'un compte Gitlab

La figure 33 illustre le processus de création et de configuration d'un compte gitlab, y compris l'intégration des différents outils utilisés pour faire fonctionner la solution et la gestion des accès.

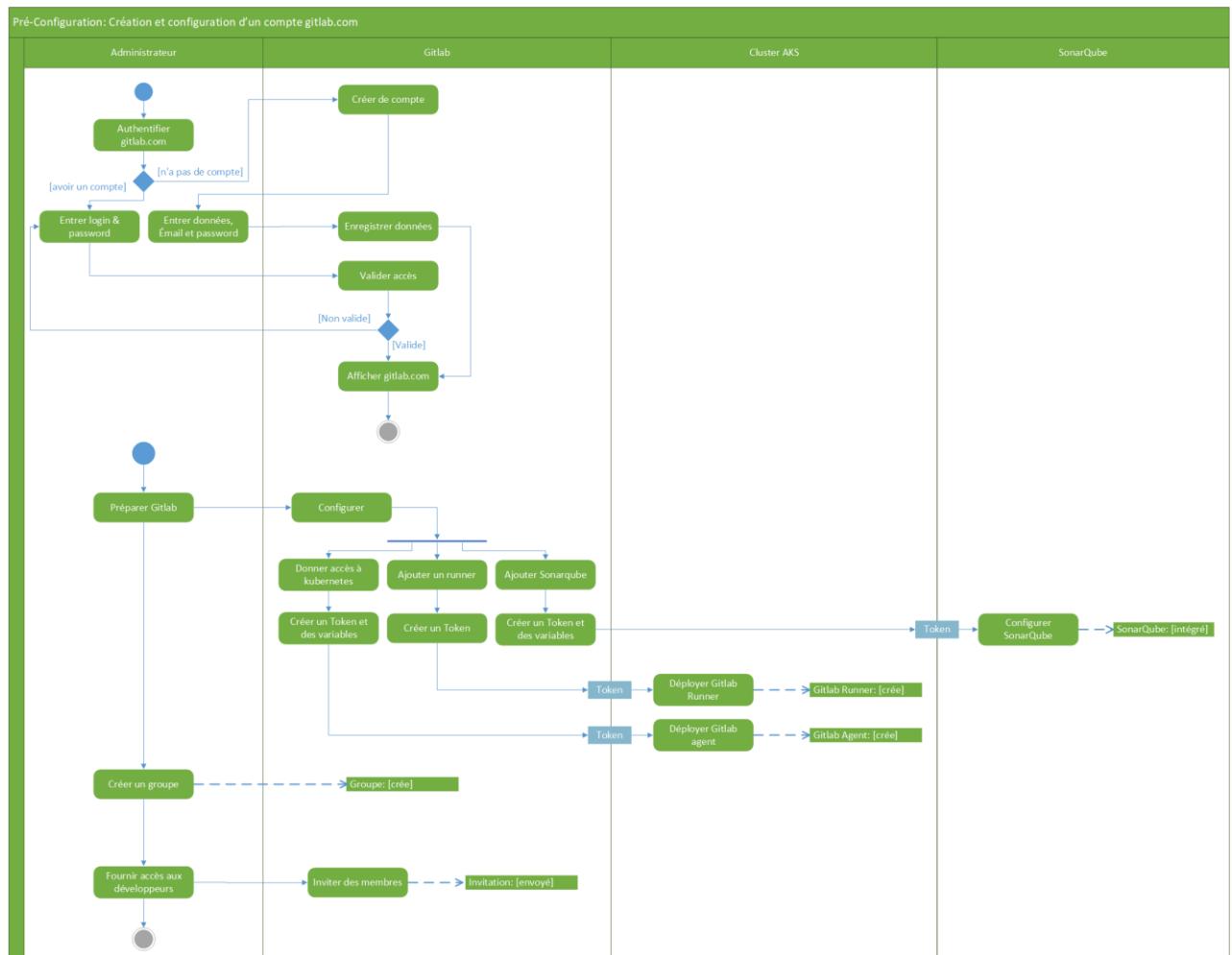


Figure 33 : Processus de création et configuration d'un compte Gitlab

#### 4.3.4.4. Processus général d'intégration continue et de déploiement du pipeline

Notre processus, qui commence lorsqu'un développeur expose son code sur Gitlab-Git et se termine lorsque les clients peuvent accéder à l'application, sera présenté dans un diagramme d'activités qui nous aidera à mieux comprendre la séquence des événements. La figure 34 montre le diagramme d'activité.

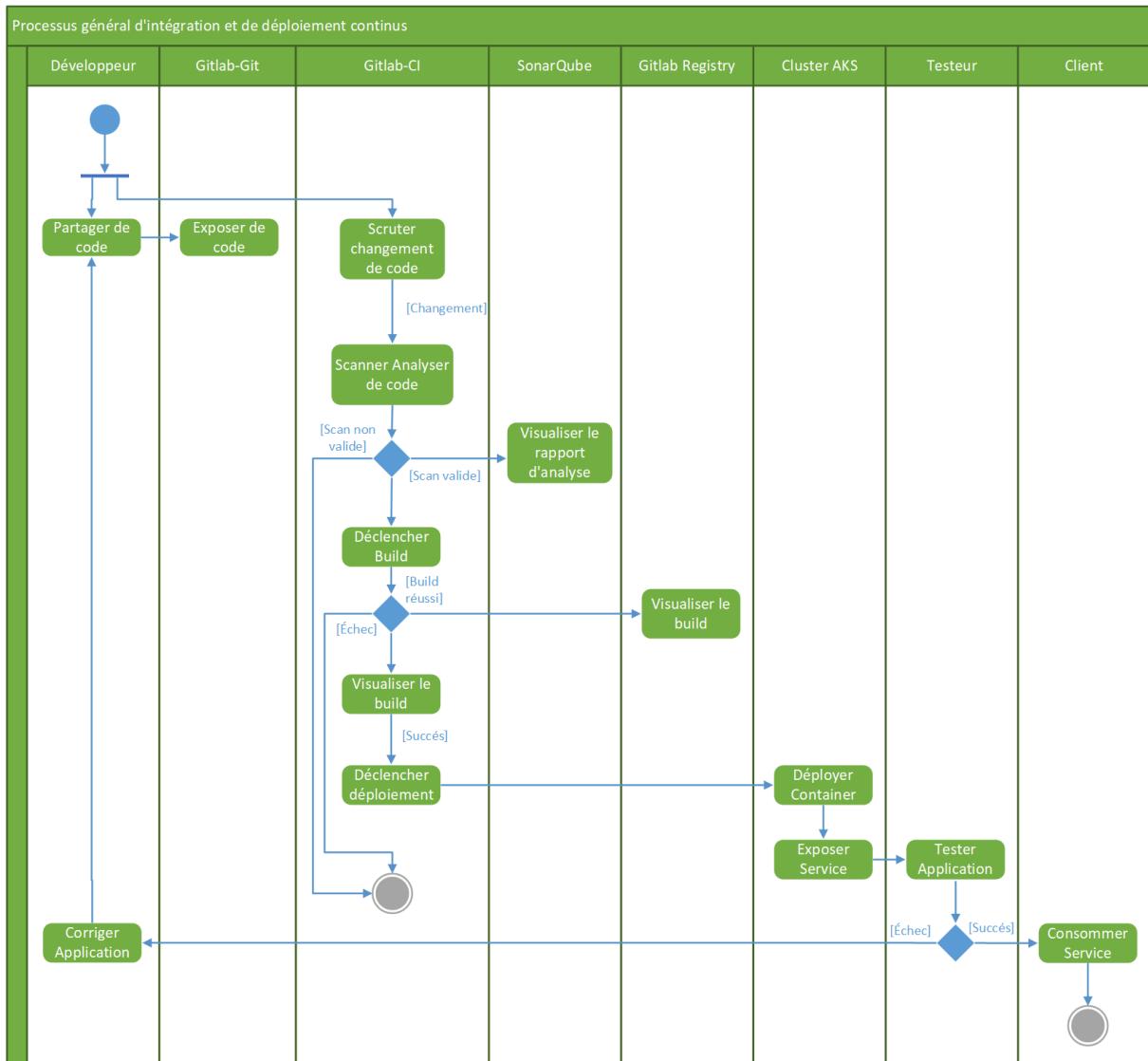


Figure 34 : Processus générale du pipeline CI/CD

Ce diagramme illustre l'implication des différents acteurs lors du déclenchement du pipeline, et les actions que chacun d'entre eux doit entreprendre dans un environnement et un ordre spécifiques, une fois que les conditions adéquates sont réunies.

Le déclenchement du pipeline se produit lorsqu'un développeur dépose une nouvelle version du code dans le référentiel Git de Gitlab.

Suite à modification dans ce référentiel, Gitlab-CI déclenche une analyse de code par le moteur d'analyse « Scanner » du SonarQube, et si l'analyse est réussie, un build est déclenché. On peut voir le rapport de l'analyse du code dans le serveur centralisé « SonarQube ».

Une fois le build et l'envoi de l'image vers le registre Gitlab terminés, un déploiement de conteneur est effectué au niveau de Kubernetes, exposant une URL qui permet aux clients d'accéder à l'application.

#### 4.4. Conclusion

Nous avons expliqué dans ce chapitre les différentes architectures génériques de solutions puis introduisons les aspects dynamiques de ces dernières. Le prochain chapitre se concentrera sur la réalisation notre travail.

# Chapitre 5

## RÉALISATION

### 5.1. Introduction

Maintenant que nous avons terminé la partie conceptuelle du projet, nous pouvons passer à la partie réalisation, où nous présentons les plateformes, les outils et l'infrastructure dans lequel nous travaillons, puis nous décrivons les technologies sur lesquelles le système s'appuie. Enfin, nous présentons des captures d'écran exprimant les résultats de différentes exécutions et déploiements effectués manuellement ou automatiquement par les tâches de notre pipeline de projet.

### 5.2. Environnement de travail

Comme nous avons vu dans l'architecture générale, notre solution s'appuie sur deux services cloud : AKS (Azure Kubernetes Service) PAAS et GitLab.com SaaS, nous citons ci-dessous les caractéristiques techniques fournies :

#### 5.2.1. Environnement Microsoft Azure :

Vous trouverez dans le tableau 27 les spécifications techniques du service Azure Kubernetes (AKS) sur Microsoft Azure.

Caractéristiques	Ressources
Nom d'instance	Standard_DS2_v2
Système d'exploitation	Linux
Mémoire	7 GB RAM
Processeur	Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz
Disque dur	1023 GB

Tableau 27 - Ressources réservées à un nœud du cluster AKS

Nous avons utilisé deux clusters avec ce type d'instance pour nos environnements de développement et de production, qui constituent nos plateformes de déploiement et d'exécution d'applications.

#### 5.2.2. Plateforme GitLab SaaS :

Pour la partie gestion de version, l'outil d'intégration et de déploiement continu et le registre d'image nous avons créé un compte sur la plateforme GitLab.com SaaS avec les caractéristiques essentielles pour les utilisateurs individuels, ci-dessous les caractéristiques techniques:

- 5GB de stockage
- 10 Go de transfert par mois
- 400 minutes de calcul par mois
- 5 utilisateurs par groupe de premier niveau

#### 5.2.3. Environnement local :

- Visual Studio Code
- Binaires installés: PgAdmin 4, HELM, Terraform, kubectl

### 5.3. Technologies utilisées

Une liste des technologies que nous utilisons pour mener à bien notre travail est présenté dans cette section.

#### 5.3.1. Configuration déclarative: YAML

YAML (Yet Another Markup Language) est un langage de sérialisation de données lisible par l'homme pour l'écriture de fichiers de configuration. Il est populaire car il est conçu pour être plus facile à comprendre que les autres langages de programmation. Pour toutes ces raisons, il est utilisé par des logiciels tels que Kubernetes ou Ansible. [14]

#### 5.3.2. Plateforme de déploiement: Kubernetes

L'objectif de notre projet est déployer et mettre en production une instance Odoo (logiciel de gestion intégré) pour un client en utilisant une méthode agile et sécurisée dans un environnement qui garantit la haute disponibilité, la tolérance aux pannes et l'accès sécurisé. À cet égard, Kubernetes fournit un cluster avec des nœuds, chacun pouvant héberger plusieurs nœuds de travail via un système de pods. Par conséquent, nous avons choisi Kubernetes comme système de gestion de cluster pour gagner plus de ressources et de flexibilité. Cependant, pour héberger le cluster Kubernetes lui-même, nous avons choisi le service managé d'Azure, Azure Kubernetes Services (AKS). AKS est une plate-forme d'hébergement fournie par Azure pour exécuter des applications dans des clusters Kubernetes sans avoir à vous soucier de configurations informatiques et réseau étendues. À l'aide d'AKS, des nœuds peuvent être ajoutés et supprimés du cluster en fonction de la charge de travail.

La version de Kubernetes réservée à notre AKS est 1.26.6.

#### 5.3.3. Technologie de configuration et de provisionnement : Terraform

Terraform, développé par HashiCorp, est un outil d'Infrastructure as Code (IaC) open source. Terraform a la capacité d'automatiser et de superviser le provisionnement de différentes

modèles du cloud IaaS, PaaS et SaaS. Il peut créer et gérer ces ressources simultanément chez plusieurs fournisseurs.

Dans notre projet, nous avons utilisé la v1.5.7 du Terraform.

### 5.3.4. Gitlab

Comme indiqué dans la phase d'étude comparative, Il s'agit d'une plateforme DevOps complète qui permet aux professionnels d'effectuer toutes les tâches d'un projet, de la planification du projet à la gestion du code source, en passant par la surveillance et la sécurité. De plus, il permet aux équipes de collaborer et de créer de meilleurs logiciels.

#### 5.3.4.1. *Les fonctionnalités principales :*

- Outil de gestion de version
- Pipeline DevOps
- Registre des conteneurs

#### 5.3.4.2. *Les fonctionnalités secondaires :*

- Événements d'audit
- Gestion de la conformité
- Authentification et autorisation
- Gestion des flux de valeur
- Suivi du temps
- Gestion et suivi des projets

#### 5.3.4.3. *Architecture Gitlab*

##### a. Architecture globale

1. Un développeur peut accéder et modifier la branche de test depuis son poste de travail.
2. Les développeurs poussent les modifications du code pour que leur code soit versionné (commit).

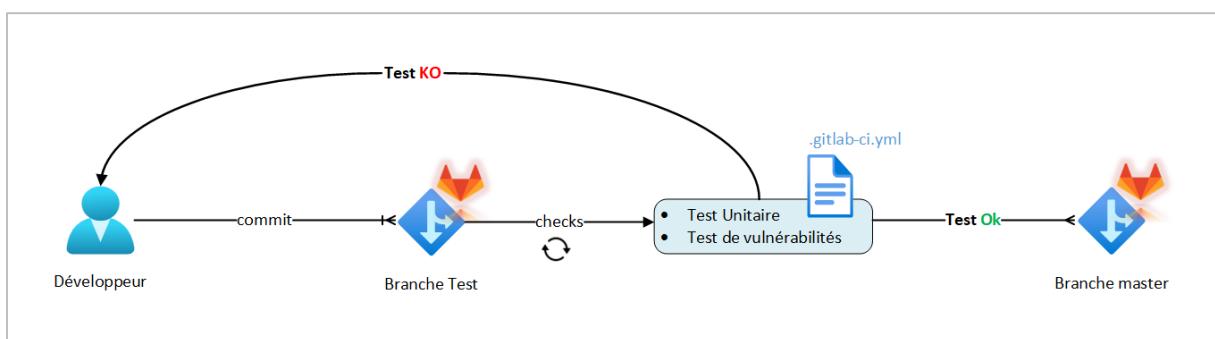


Figure 35 : Architecture globale du Gitlab

3. Après la soumission, le pipeline sera déclenché, y compris les tests unitaires et les tests de vulnérabilité.
4. Si le test réussit, les modifications sont également déployées dans la branche master, sinon les modifications sont annulées, aucune modification n'est validée dans la branche master, et le développeur doit vérifier son code pour ajuster ses tickets.

Le diagramme d'activité de la figure 36 représente l'exécution d'une tâche standard nommée « Test » dans un pipeline Gitlab CI.

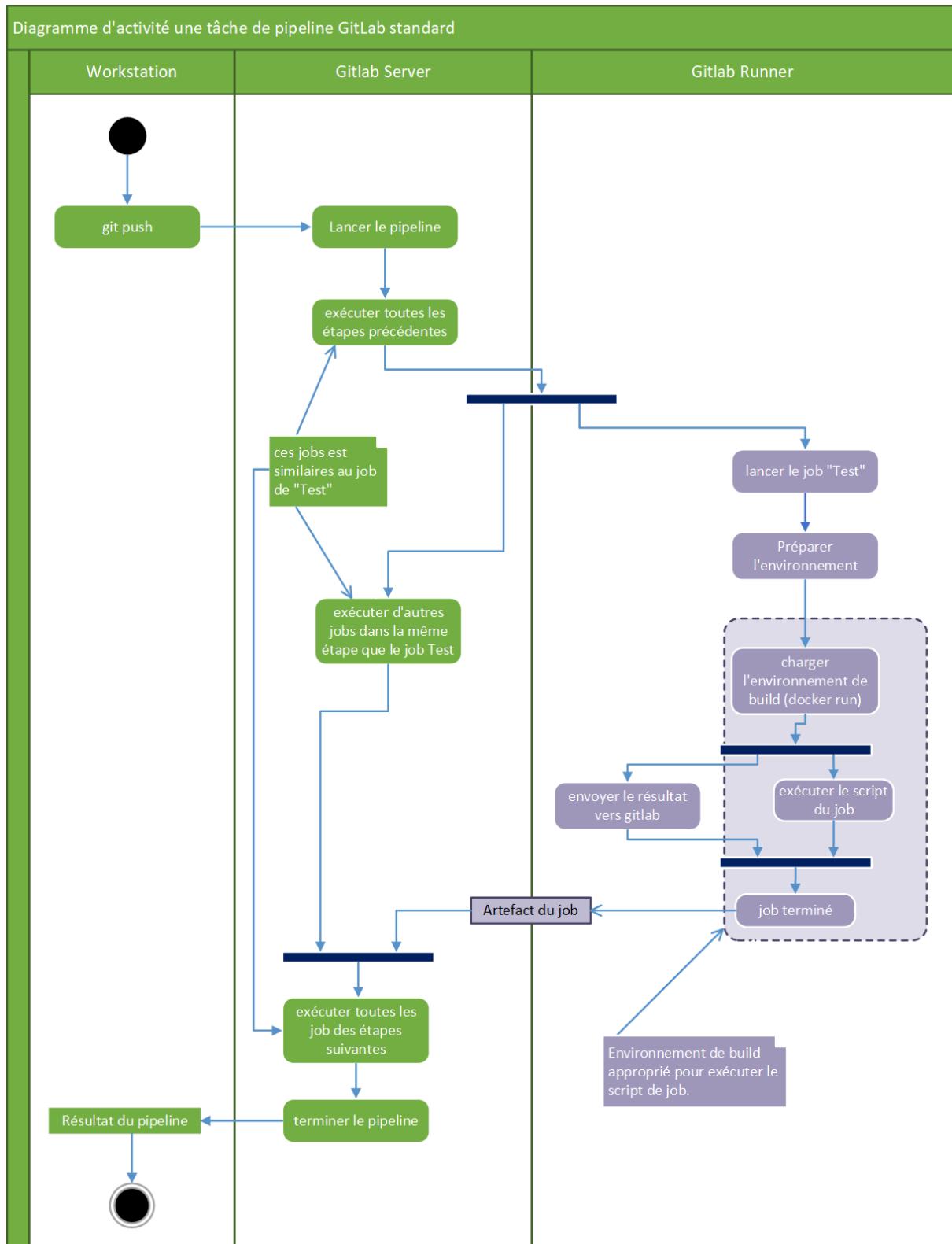


Figure 36 : Processus d'une tâche de pipeline GitLab standard

### b. Architecture d'intégration continue

GitLab CI/CD est un outil puissant intégré à GitLab qui vous permet d'appliquer toutes les pratiques d'intégration et de déploiement continus (CI/CD) à votre logiciel sans avoir besoin d'applications ou d'intégrations tierces.

Le code utilisé dans l'environnement est stocké dans les référentiels Git de GitLab. Les développeurs poussent les changements de code. Pour chaque poussée vers le référentiel, vous pouvez créer un ensemble de scripts pour créer et tester automatiquement votre application, réduisant ainsi le risque d'introduire des bogues dans votre application.

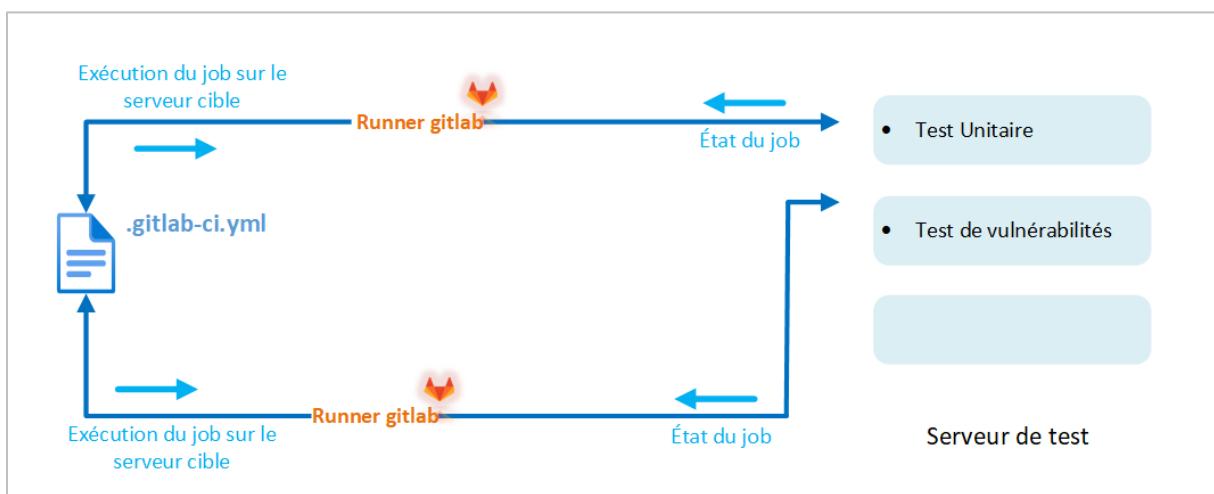


Figure 37 : Processus d'intégration continue CI

Cette pratique est appelée intégration continue, et pour chaque changement apporté à l'application (même dans la branche de développement), la tâche est automatiquement exécutée en continu, garantissant que les changements introduits réussissent tous les tests, directives et conformité du code que vous avez établi pour la norme d'application.

### c. Architecture de déploiement continu

Le déploiement continu est également une étape d'intégration continue, similaire à la livraison continue. La différence est qu'au lieu de déployer manuellement l'application, vous la configurez pour qu'elle se déploie automatiquement. Le déploiement de l'application ne nécessite aucune intervention manuelle.

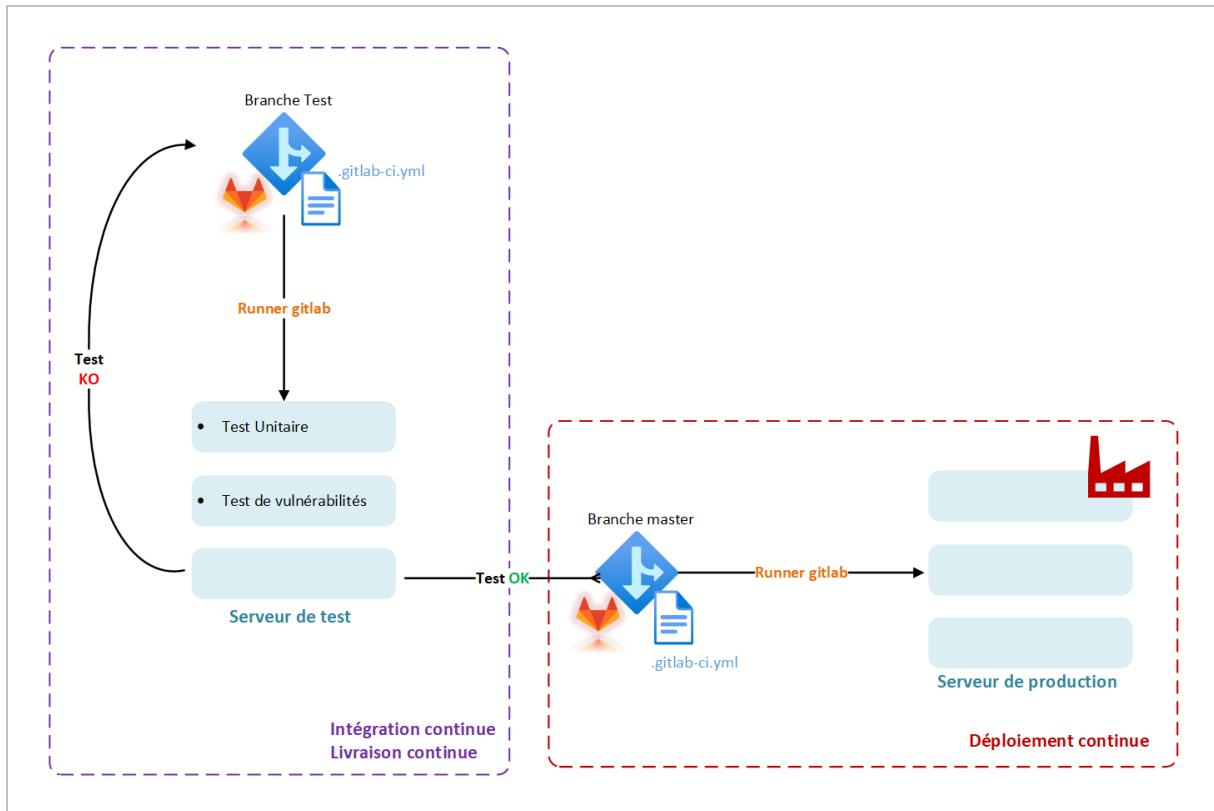


Figure 38 : Processus de déploiement continu CD

Une fois les tests validés dans la partie intégration continue, les modifications sont déployées dans la branche master et le serveur de production, à l'aide de l'agent gitlab runner installé dans le serveur de test et du fichier ".gitlab-ci.yml" intégré dans la branche master.

### 5.3.5. Serveur d'application

#### 5.3.5.1. Le progiciel de gestion Odoo :

Odoo, autrefois connu sous le nom d'OpenERP, est un éditeur de logiciels open source fondé en 2004 qui propose une suite de modules de gestion d'entreprise entièrement intégrés.

Odoo est le programme de gestion d'entreprise le plus évolutif et le plus largement installé au monde, avec des applications pour répondre à tous les besoins d'une entreprise, de la gestion de la relation client à la création de sites Web et de courriers électroniques, en passant par le commerce, la production, la gestion des stocks, la comptabilité, le tout parfaitement intégré.  
[15]

Dans notre projet, nos déploiements ont été effectués avec la version 16.

#### 5.3.5.2. PostgreSQL :

PostgreSQL est un système de gestion de bases de données relationnelles puissant, open source et orienté objet qui prend en charge en toute sécurité les charges de travail de données

les plus complexes. MySQL donne la priorité à l'évolutivité et aux performances, tandis que PostgreSQL donne la priorité à la conformité et à l'évolutivité SQL. [16]

La version que nous avons utilisée est 13.

#### 5.3.5.3. PgAdmin

pgAdmin fait partie du logiciel de gestion de base de données open source utilisé pour gérer les bases de données PostgreSQL 7.3 et versions ultérieures. C'est un peu comme Access de Microsoft. Il comprend une interface administrateur et vous permet d'écrire des requêtes SQL simples et complexes. Un éditeur de code de programme est également intégré.

La version utilisée est la 4, vous pouvez utiliser pgAdmin 4 comme runtime de bureau ou sur un serveur d'applications web, selon votre cas d'utilisation. [17]

#### 5.3.6. Outil de test : Sonarqube

Comme nous l'avons vu dans l'étude comparative, Sonarqube est un outil très puissant il va nous permettre d'analyser notre code de manière statique. C'est-à-dire via une analyse à base d'expression régulière. Sonarqube nous donne un retour d'information spécifique sur les paramètres de code ci-dessous:

- Bugs
- Vulnérabilités de sécurité
- Code smell « mauvaise odeur » et dette technique (maintenabilité)
- Duplications de code

#### 5.3.7. Outil de build d'image docker : Kaniko

Kaniko est un mécanisme de construction qui crée et transmet des images de conteneurs vers le registre sans utiliser Docker.

La création d'images Docker peut être effectuée dans des conteneurs ou des clusters Kubernetes. Étant donné que Kaniko n'a pas besoin de Docker pour créer des images, la construction peut être effectuée en toute sécurité et faire partie du processus de déploiement. [18]

### 5.4. Principaux développements

Pour illustrer les principaux développements, nous présentons des captures d'écran décrivant la mise en œuvre des fonctionnalités du système, notamment l'exécution de notre pipeline, l'affichage des résultats des jobs, la configuration multiplateforme et multiprocessus, la surveillance et la pré-configuration.

### 5.4.1. Intégration et déploiement continus

Nous montrons ci-dessous l'exécution de notre pipeline CI/CD composé de 5 jobs, la première phase est l'intégration et la livraison continue, qui consiste en 4 jobs (analyse du code, construction d'une image, livraison et test), le déploiement continu est la deuxième phase qui consiste en une seule job qui est le déploiement dans l'environnement de production.

La figure 39 illustre le processus d'intégration de la livraison continue, qui est exécuté par Gitlab-CI à la suite d'une modification du référentiel Git dans la branche « main » dédiée au développement dans notre cas.

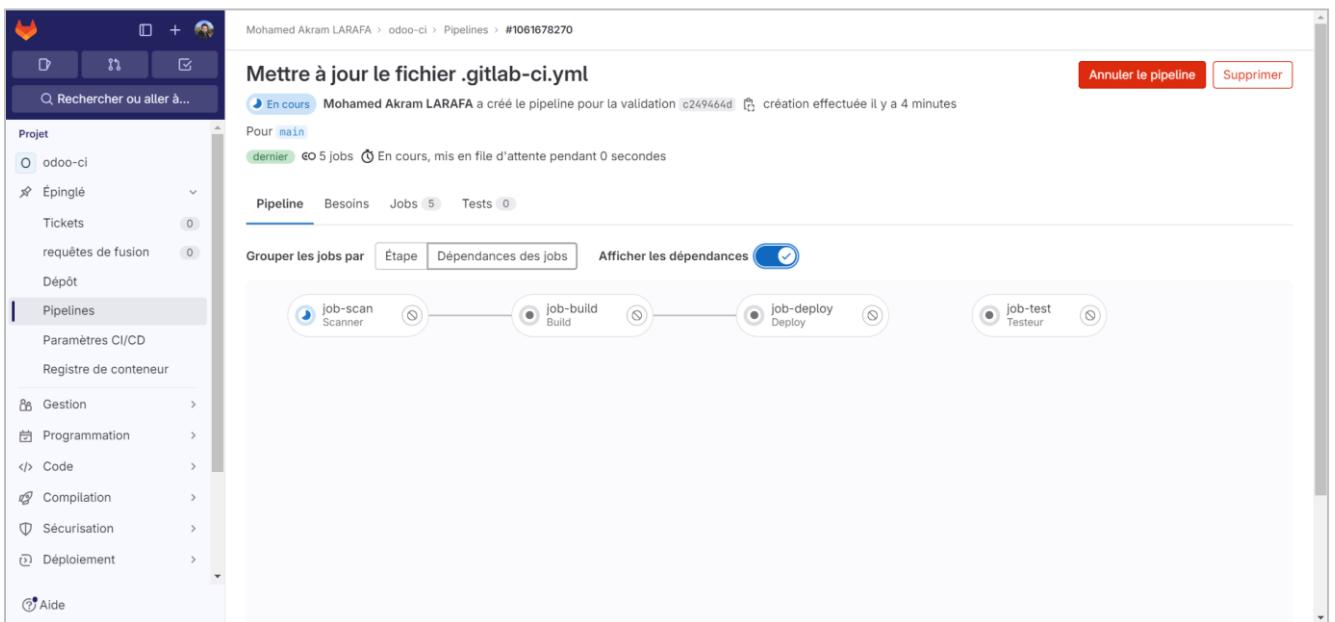


Figure 39 : Processus d'intégration et livraison continue

La figure 40 illustre le processus de déploiement continu, qui est exécuté par Gitlab-CI à la suite d'une fusion du code dans le référentiel Git de la branche « main » vers la branche « production » dédiée à la production dans notre cas.

## Chapitre 5. Réalisation

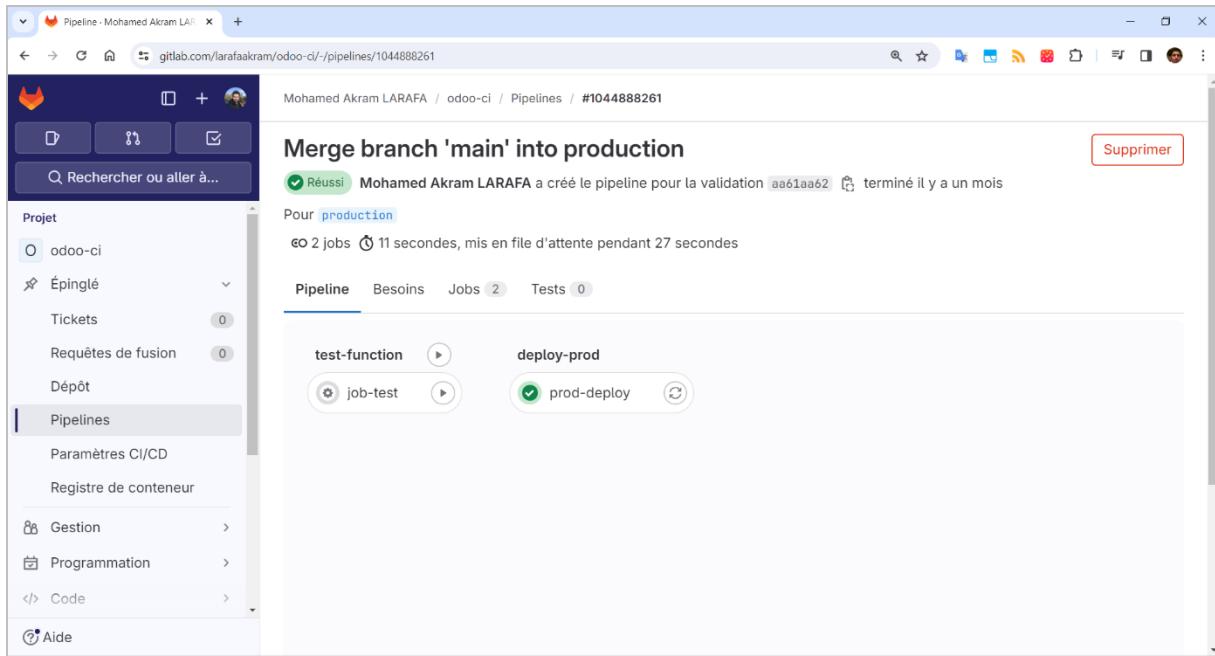


Figure 40 : Processus de déploiement continu

La figure 41 montre le résultat du scan et de l'analyse déclenchés par Gitlab-CI et effectués par Sonarqube.

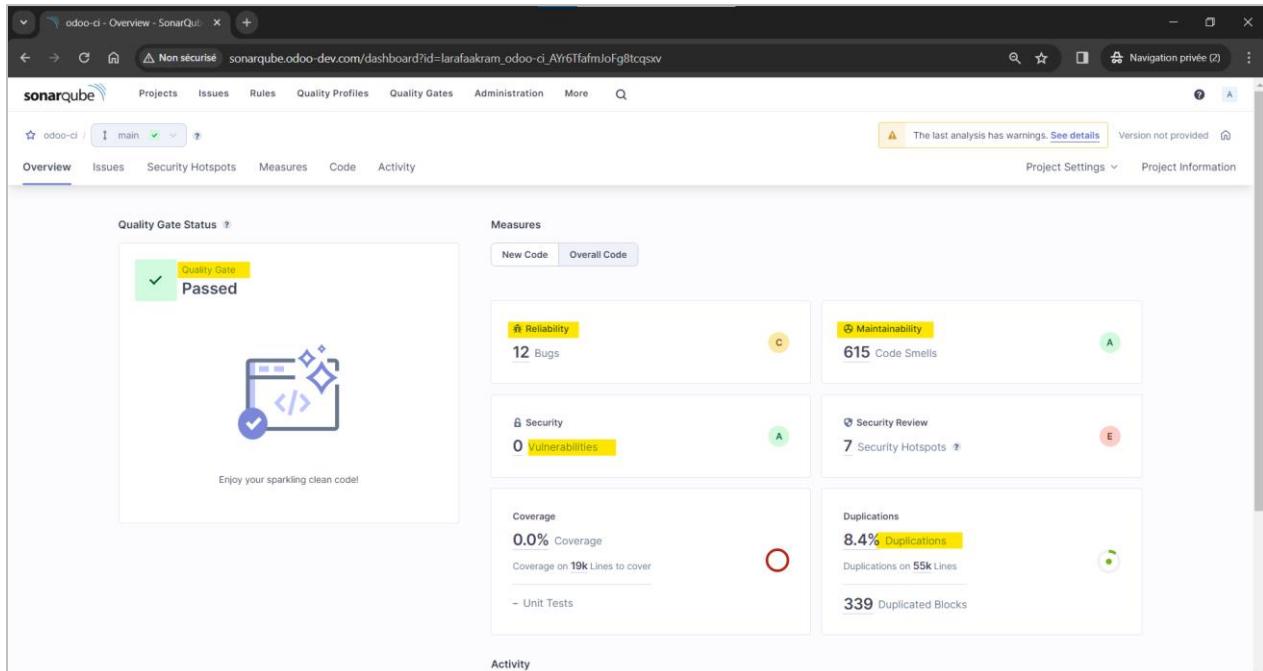


Figure 41 : Résultat du scan et analyse du code source

La figure 42 montre le résultat d'exécution du job de Build de l'image et leur envoi vers le registre Gitlab, remarquez le numéro d'empreinte de l'image marqué en jaune.

## Chapitre 5. Réalisation

The screenshot shows the GitLab CI interface for a project named 'odoo-ci'. The left sidebar shows various project sections like 'Projet', 'Jobs' (which is selected), 'Éditeur de pipeline', and 'Planifications de pip...'. The main area displays the execution log for job #5344442973. The log output is as follows:

```

55 INFO[0058] Util.Lookup returned: &{Uid: HomeDir:/root}
56 INFO[0058] Performing slow lookup of group ids for root
57 INFO[0058] Running: [/bin/bash -xo pipefail -c chmod +x /entrypoint.sh]
58 + chmod +x /entrypoint.sh
59 INFO[0058] Taking snapshot of full filesystem...
60 INFO[0060] USER odoo
61 INFO[0060] Cmd: USER
62 INFO[0060] ENTRYPOINT ["/entrypoint.sh"]
63 INFO[0060] CMD ["odoo", "-i base"]
64 INFO[0060] Pushing image to registry.gitlab.com/larafaakram/odoo-ci:sha256:3b772c06c6b79b716407cb1c1f2a5417352d1ddcac67a6b6cb63862e292b0618
65 INFO[0079] Pushed registry.gitlab.com/larafaakram/odoo-ci@sha256:3b772c06c6b79b716407cb1c1f2a5417352d1ddcac67a6b6cb63862e292b0618
66 Cleaning up project directory and file based variables
67 Job succeeded

```

On the right side, there are details about the job: Duration (1 minute 30 seconds), Status (Terminé), Pipeline (Pipeline #1044823061), and Task (job-build). Below the log, a section titled 'Tâches liées' shows a link to 'job-build'.

Figure 42 : Résultat de l'exécution du job de Build

La figure 43 montre la liste des images créées précédemment par les tâches de Build qui sont hébergées dans le registre Gitlab. L'image marquée en jaune est la même image que celle montrée dans la tâche de Build.

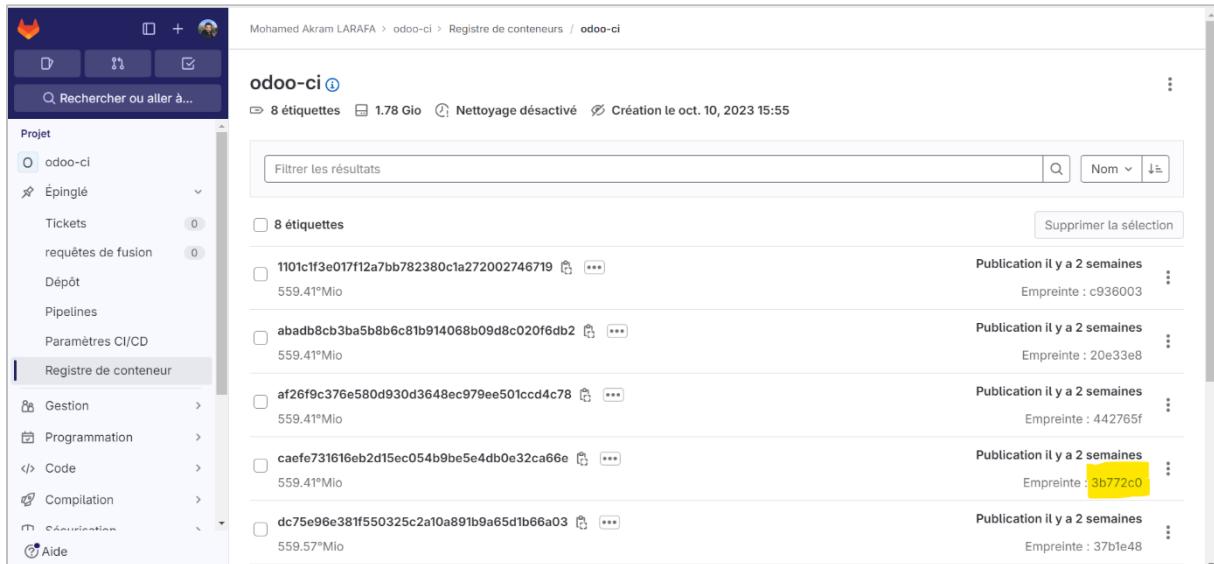


Figure 43 : Liste des images créées et hébergées dans le registre Gitlab

La figure 44 montre la liste des applications déployées, dont le déploiement « dev-odoo-deployment » a été effectué par le job « Deploy » dans l'environnement de développement.

The screenshot shows the Portainer.io interface. On the left, there is a sidebar with various options like Home, local, Dashboard, Custom Templates, Namespaces, Helm, Applications (which is selected), Services, Ingresses, ConfigMaps & Secrets, Volumes, and Cluster. Below these are Settings sections for Users, Environments, Registries, Authentication logs, and Notifications. At the bottom of the sidebar is the Portainer.io Community Edition 2.18.1 logo.

The main area is titled "Applications" and "Application list". It displays a table of deployed applications:

Name	Stack	Namespace	Image	Application Type	Status	Published	Created
dev-odoo-deployment	external	-	odoodev	Deployment	Replicated 1 / 1	Yes	2023-10-20 16:54:28
dev-pgadmin	external	-	odoodev	StatefulSet	Replicated 1 / 1	Yes	2023-10-07 23:04:42
Published URL(s)	<a href="http://pgadmin.odoo-dev.com/">http://pgadmin.odoo-dev.com/</a>						
dev-team-postgresql	external	-	odoodev	Helm	Ready	Yes	2023-10-07 22:49:47
Name	Stack	Namespace	Image	Application Type	Status	Published	Created
dev-team-postgresql	external	-	odoodev	StatefulSet	Replicated 1 / 1	Yes	2023-10-07 22:49:47

At the bottom right of the main area, it says "Items per page" with a dropdown set to 25.

Figure 44 : Déploiement de l'application dans l'environnement de développement

### 5.4.2. Configuration

Pour déployer correctement une application sur Kubernetes, le module de configuration est requis. Comme nous l'avons défini dans la section « Configuration » de la phase « Représentation des besoins », plusieurs éléments sont nécessaires au succès de l'exécution de notre pipeline et notre déploiement.

#### 5.4.2.1. Déploiement et configuration du Portainer

Portainer est l'interface graphique que nous avons utilisée pour gérer Kubernetes dans notre environnement de développement, et la commande ci-dessous nous aide à l'installer.

```
# helm install --create-namespace -n portainer portainer/portainer --values values.yaml
```

La Figure 45 montre un tableau de bord regroupe des éléments gérés par Portainer, avec un menu à gauche pour d'autres objets tel que Helm, Ingress et Clusters.

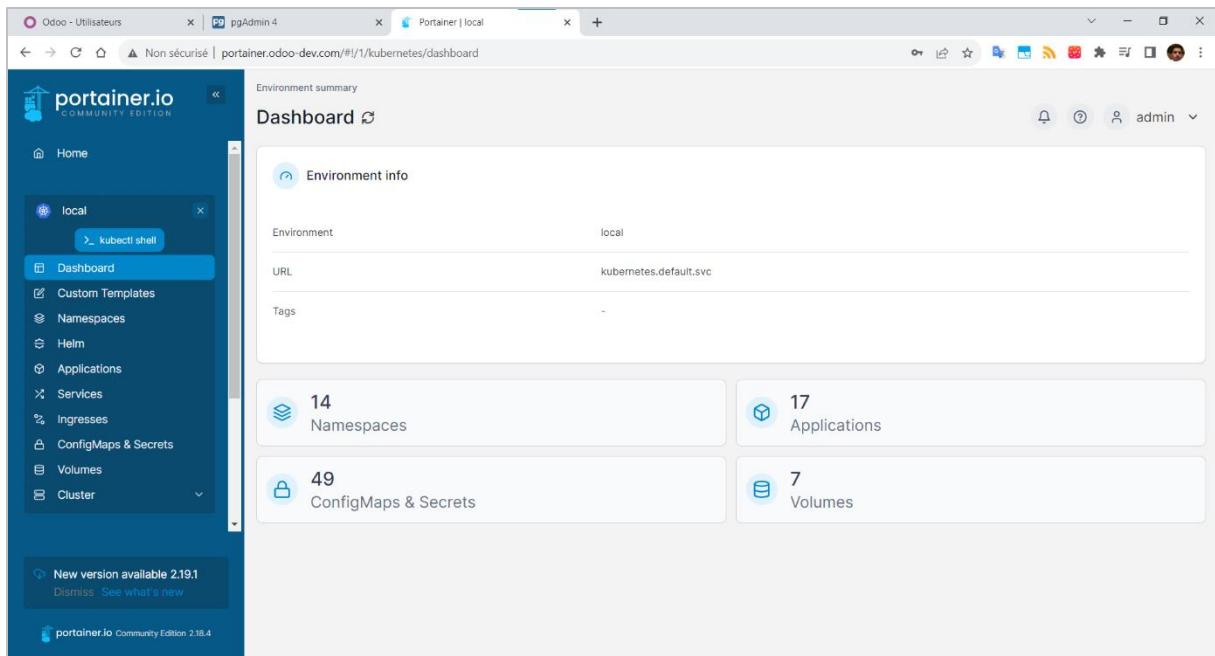


Figure 45 : Dashboard du Portainer

### 5.4.2.2. Déploiement et configuration du Ingress-Controller

Dans notre AKS de production utilisant NGINX comme proxy inverse et équilibrer de charge, nous allons déployer ingress-nginx qui est un contrôleur Ingress pour Kubernetes avec la commande ci-dessous :

```
# helm install ingress-nginx ingress-nginx/ingress-nginx --namespace ingress-nginx --set controller.replicaCount=1 --set controller.nodeSelector."beta\\.kubernetes\\.io/os"=linux --set defaultBackend.nodeSelector."beta\\.kubernetes\\.io/os"=linux --set controller.service.externalTrafficPolicy=Local --set --create-namespace
```

La figure 46 montre le résultat de notre déploiement :

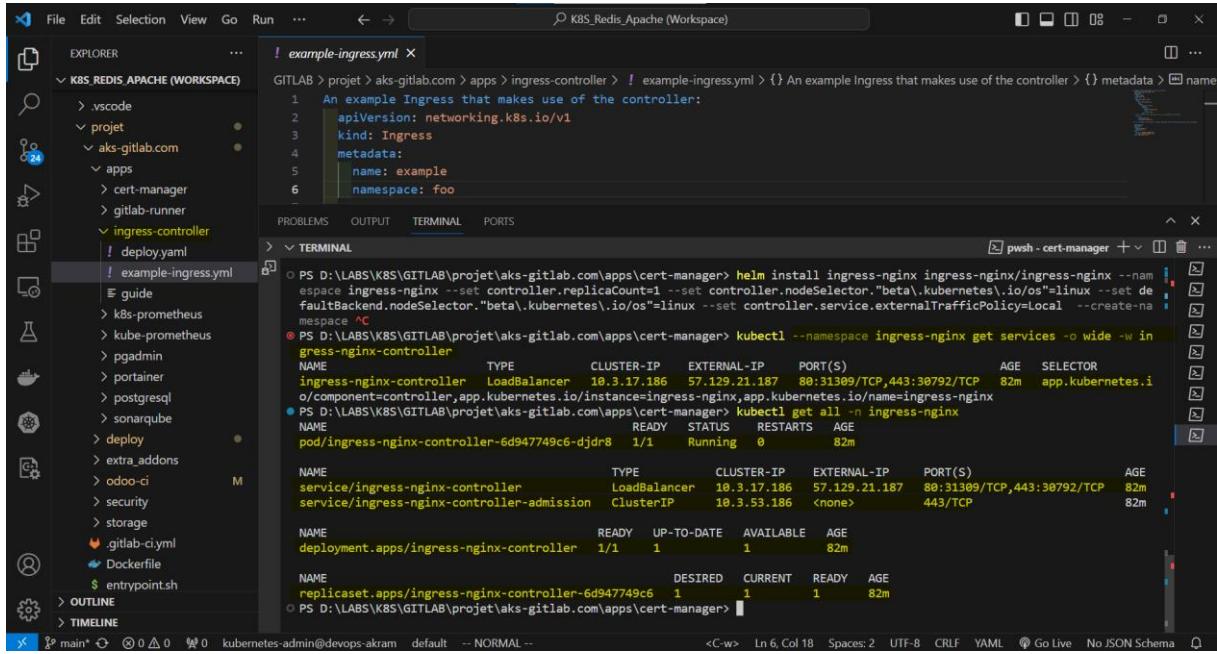


Figure 46 : Déploiement et configuration du Ingress-Controller

#### 5.4.2.3. Déploiement et configuration du Cert-Manager

Pour établir une connexion sécurisée entre odoo intances et les clients finaux en utilisant des certificats SSL/TLS et l'extension HTTPS, nous avons choisi le contrôleur Cert-Manager pour la gestion des certificats. Nous allons mettre en place un Cert-Manager avec l'émetteur Lets Encrypt.

La commande ci-dessous permet de faire le déploiement du Cert-manager :

```
# helm install cert-manager cert-manager/cert-manager --namespace cert-manager --create-namespace --set installCRDs=true
```

La commande ci-dessous est utilisé pour déployer l'émetteur Let's Encrypt, qui représente l'autorité de certification (CA) capables de générer des certificats signés :

```
# kubectl apply -f letsencrypt-prod.yaml
```

La figure 47 montre le manifeste de notre émetteur ou « clusterIssuer » :

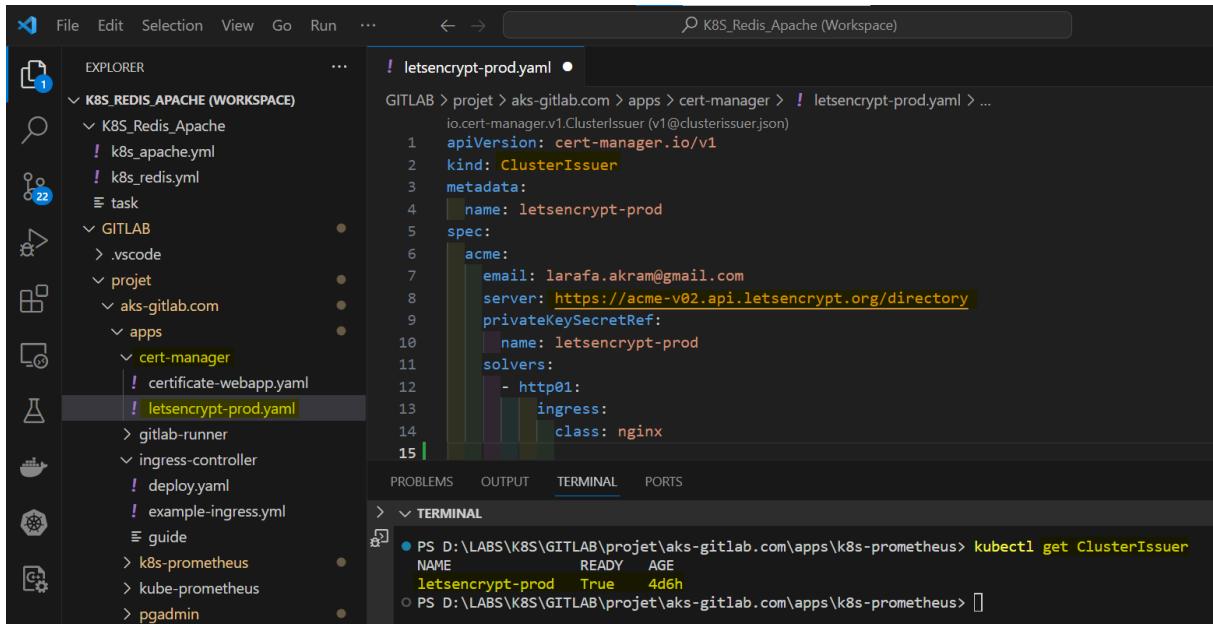


Figure 47 : Déploiement d'émetteur Lets Encrypt

#### 5.4.2.4. Déploiement et configuration de Sonarqube

Sonarqube est l'outil que nous avons utilisé pour scanner et analyse notre code statique, ci-dessous la commande d'installation :

```
# helm install -n sonarqube sonarqube sonarqube/sonarqube --create-namespace --values values.yaml
```

La figure 48 montre les ressources créées suite à l'installation de Sonarqube dans notre environnement de développement.

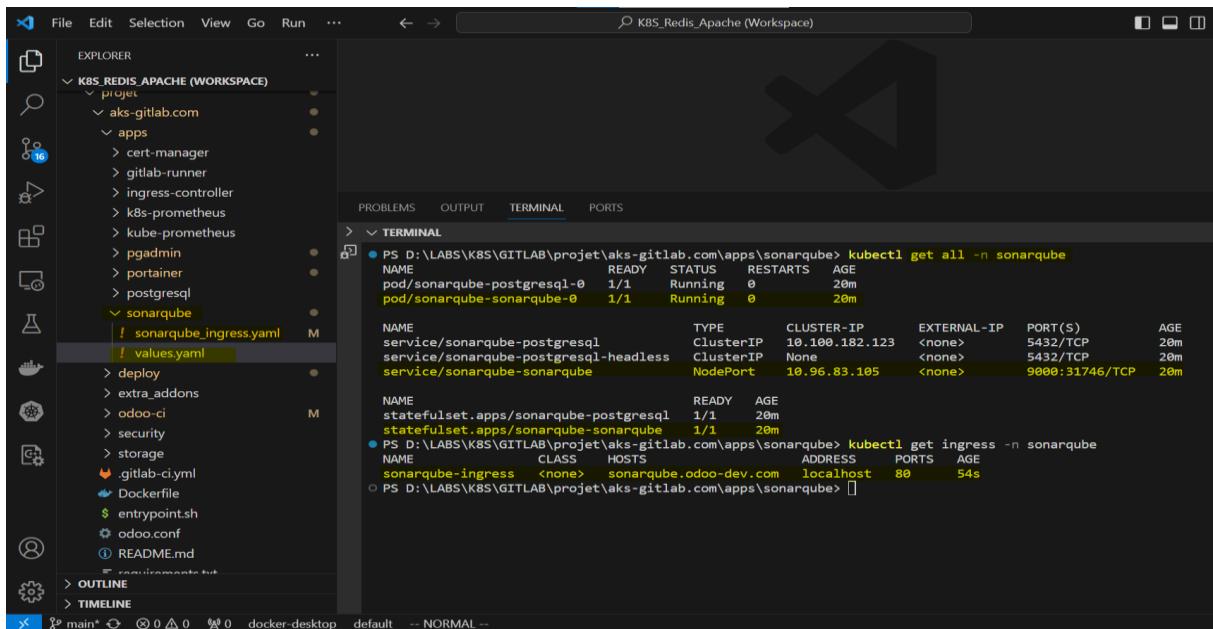


Figure 48 : Déploiement et configuration de Sonarqube

Sonarqube interface showing the import of GitLab projects. The list includes:

- DockerTuto Mohamed Akram LARAFA
- Odoo Mohamed Akram LARAFA
- demo Mohamed Akram LARAFA
- odo-ci Mohamed Akram LARAFA
- odo-v2 Mohamed Akram LARAFA

Buttons for each item include "See on GitLab" and "Import". Status indicators show some are already imported.

Figure 49 : Sonarqube intégré avec Gitlab

### 5.4.2.5. Dockerfile

La Figure 50 montre la définition de notre Dockerfile responsable de la création de l'image au niveau du pipeline Job Build ou via la console.

Notre image sera basée sur la version 16 d'Odoo.

```

FROM odoo:16.0
# Copy entrypoint script and Odoo configuration file
COPY ./entrypoint.sh /
COPY ./odoo.conf /etc/odoo/
COPY ./requirements.txt /etc/odoo/
COPY ./extra-addons /mnt/extras-addons
COPY wait-for-psql.py /usr/local/bin/wait-for-psql.py
# Expose Odoo services
EXPOSE 8069 8071 8072
# Set the default config file
ENV ODOO_RC /etc/odoo/odoo.conf
USER root
RUN chmod +x /usr/local/bin/wait-for-psql.py
RUN chmod +x /entrypoint.sh
RUN pip3 install -r /etc/odoo/requirements.txt
# Set default user when running the container
USER odoo
#RUN chown -R odoo:odoo /var/lib/odoo
ENTRYPOINT ["/entrypoint.sh"]
CMD ["odoo", "-i base"]

```

Figure 50 : Définition d'une Dockerfile

### 5.4.2.6. Crédit et configuration d'un ServiceAccount

La Figure 51 montre les éléments utilisés pour garantir un accès sécurisé à nos clusters (Namespaces, docker registry, ClusterRole, ServiceAccount) à partir des plateformes tierces, Gitlab-CI dans notre cas.

The screenshot shows the VS Code interface with the title bar "K8S.Redis\_Apache (Workspace)". The left sidebar is the Explorer view, showing a project structure under "K8S.REDIS\_APACHE (W...)". The "gitlab-prod-rolebinding.yaml" file is open in the main editor area. The terminal tab is active, displaying several commands run in a PowerShell window titled "pwsh - aks-gitlab.com". The commands are related to Kubernetes resources like secrets, service accounts, and rolebindings.

```

PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com> kubectl get -l project=devops sa
NAME      SECRETS   AGE
gitlab    0          83m
PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com> kubectl get -l project=devops secret
NAME      TYPE      DATA   AGE
gitlab-secret  kubernetes.io/service-account-token  3       83m
PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com> kubectl get -l project=devops ClusterRole
NAME      CREATED AT
deploy-role  2023-11-06T17:32:55Z
nginx-ingress  2023-11-06T17:32:55Z
volumes-access  2023-11-06T17:32:55Z
PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com> kubectl get -l project=devops Rolebinding -n odoo-dev
NAME      ROLE      AGE
nginx-ingress  ClusterRole/nginx-ingress  82m
rb-odoo-dev-1  ClusterRole/volumes-access  82m
rb-odoo-dev-2  ClusterRole/deploy-role  82m
PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com> kubectl get -l project=devops Rolebinding -n odoo-prod
NAME      ROLE      AGE
nginx-ingress  ClusterRole/nginx-ingress  82m
rb-prod-1    ClusterRole/volumes-access  82m
rb-prod-2    ClusterRole/deploy-role  82m
PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com>

```

Figure 51 : Configuration d'accès pour un ServiceAccount

#### 5.4.2.7. Configuration et déploiement de PostgreSQL

La commande ci-dessous nous permet de déployer PostgreSQL en utilisant le gestionnaire de paquets Helm et l'application de la configuration créée dans le fichier "values.yaml".

```
# helm install prod-team-postgresql -n odoo-prod bitnami/postgresql --values values.yaml
```

Cette commande est exécutée pour nos deux environnements (odoo-prod, odoo-dev).

La figure 52 montre les ressources créées dans notre environnement de production lorsque PostgreSQL est déployé.

The screenshot shows a terminal window within a Kubernetes workspace named "K8S.REDIS\_APACHE (Workspace)". The workspace contains several projects and applications, including "aks-gitlab.com" which has "apps" like "cert-manager", "gitlab-runner", "ingress-controller", "k8s-prometheus", "kube-prometheus", "pgadmin", "portainer", and "postgresql". The "postgresql" application is currently selected.

In the terminal, the command `kubectl get all -n odoo-prod` is run, displaying the following pod and service information:

NAME	READY	STATUS	RESTARTS	AGE
pod/prod-odoo-deployment-5dd6f98744-9nwz	1/1	Running	0	14h
pod/prod-team-postgresql-0	1/1	Running	0	23h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/prod-odoo-service	NodePort	10.3.174.248	<none>	8069:30303/TCP	16h
service/prod-team-postgresql	NodePort	10.3.231.27	<none>	5432:32473/TCP	23h
service/prod-team-postgresql-h1	ClusterIP	None	<none>	5432/TCP	23h

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/prod-odoo-deployment	1/1	1	1	16h

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/prod-odoo-deployment-54f7d9989f	0	0	0	15h
replicaset.apps/prod-odoo-deployment-5b6d445d4f	0	0	0	16h
replicaset.apps/prod-odoo-deployment-5dd6f98744	1	1	1	14h
replicaset.apps/prod-odoo-deployment-7d4ddd4b55	0	0	0	15h
replicaset.apps/prod-odoo-deployment-95b65ddb5	0	0	0	15h

NAME	READY	AGE
statefulset.apps/prod-team-postgresql	1/1	23h

PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com\apps\postgresql> [REDACTED]

Figure 52 : Déploiement et configuration de PostgreSQL

#### 5.4.2.8. Déploiement et configuration de pgAdmin

La figure 53 montre l'installation de pgAdmin dans l'environnement de production à l'aide de l'outil « Kustomize », qui facilite nos déploiements dans plusieurs environnements sans qu'il soit nécessaire de modifier les manifestes d'origine.

## Chapitre 5. Réalisation

```

PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com\apps\pgadmin\overlay\prod> kubectl apply -k . -n odoo-prod
configmap/prod-pgadmin-config created
service/prod-pgadmin-service created
persistentvolume/prod-pgadmin-pv-data created
persistentvolumeclaim/prod-pgadmin-data-claim created
statefulset.apps/prod-pgadmin created
● PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com\apps\pgadmin\overlay\prod> kubectl get all -n odoo-prod
NAME                                         READY   STATUS    RESTARTS   AGE
pod/prod-odoo-deployment-5dd6f98744-9nwzs   1/1     Running   0          15h
pod/prod-pgadmin-0                           1/1     Running   0          32s
pod/prod-team-postgresql-0                   1/1     Running   0          23h

NAME                                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/prod-odoo-service   NodePort   10.3.174.248 <none>       8069:30303/TCP  16h
service/prod-pgadmin-service   NodePort   10.3.11.58   <none>       80:31018/TCP   32s
service/prod-team-postgresql   NodePort   10.3.231.27 <none>       5432:32473/TCP  23h
service/prod-team-postgresql-hl ClusterIP None           <none>       5432/TCP        23h

NAME                               READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prod-odoo-deployment 1/1     1           1          16h

NAME                               DESIRED  CURRENT  READY   AGE
replicaset.apps/prod-odoo-deployment-54f7d9989f  0        0        0      15h
replicaset.apps/prod-odoo-deployment-5b6d445d4f  0        0        0      16h
replicaset.apps/prod-odoo-deployment-5dd6f98744  1        1        1      15h
replicaset.apps/prod-odoo-deployment-7d4dd4b55   0        0        0      15h
replicaset.apps/prod-odoo-deployment-95b65ddb5  0        0        0      15h

NAME                               READY   AGE
statefulset.apps/prod-pgadmin      1/1     32s
statefulset.apps/prod-team-postgresql 1/1     23h

```

Ln 16, Col 39 Spaces: 2 UTF-8 CRLF YAML

Figure 53 : Déploiement et configuration de pgAdmin

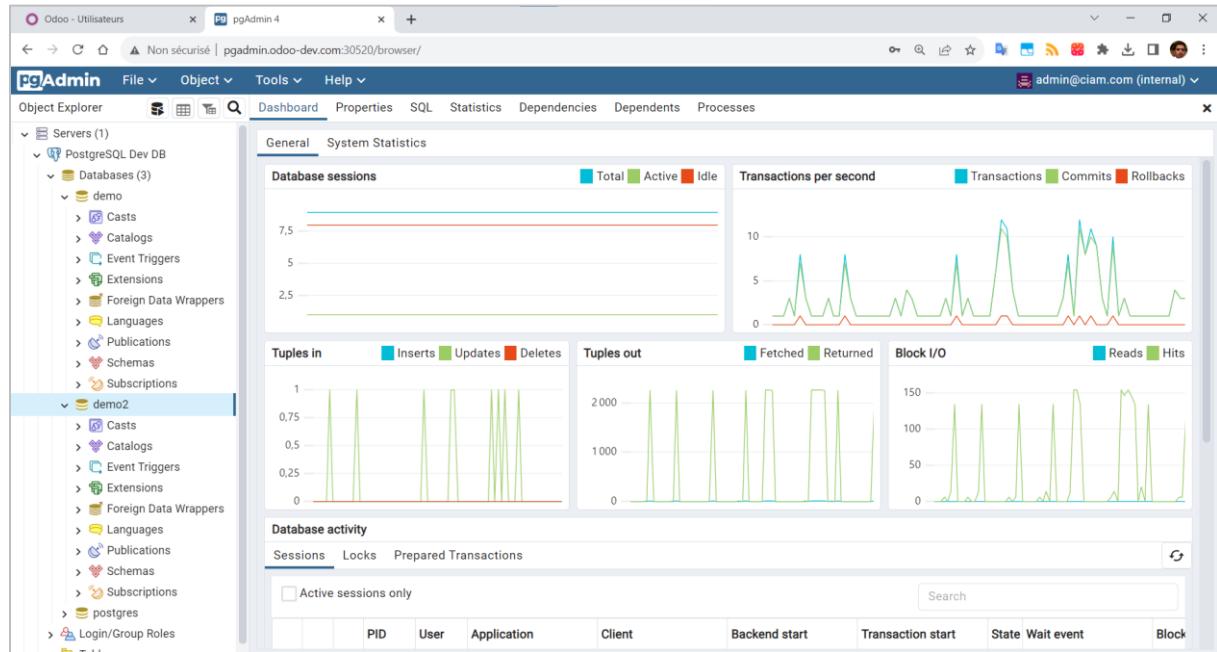


Figure 54 : Dashboard de pgAdmin 4

### 5.4.2.9. Déploiement et configuration d'instance Odoo

La Figure 51 montre le déploiement d'une instance Odoo à partir d'un Template défini avec l'outil « kustomize », qui illustre également les différents objets nécessaires au bon fonctionnement de l'application.

## Chapitre 5. Réalisation

Les types d'objets : Deployment, Service, PersistentVolumeClaim, Ingress, Certificate

```

K8S REDIS_APACHE (Workspace)
File Edit Selection View Go Run ...
TERMINAL
PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com\deploy\overlay\odoo-prod> kubectl get all -n odoo-prod
NAME           READY   STATUS    RESTARTS   AGE
pod/prod-odoo-deployment-5dd6f98744-9nwz   1/1     Running   0          16h
pod/prod-pgadmin-0   1/1     Running   0          45m
pod/prod-team-postgresql-0   1/1     Running   0          25h

NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/prod-odoo-service  NodePort   10.3.174.248 <none>       8069:30383/TCP  17h
service/prod-pgadmin-service  NodePort   10.3.82.174 <none>       80:38520/TCP   45m
service/prod-team-postgresql  NodePort   10.3.231.27 <none>       5432:32473/TCP  25h
service/prod-team-postgresql-hl ClusterIP  None            <none>       5432/TCP      25h

NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/prod-odoo-deployment  1/1     1           1           17h

NAME           DESIRED  CURRENT  READY   AGE
statefulset.apps/prod-pgadmin   1/1     0         0       16h
statefulset.apps/prod-team-postgresql  1/1     25h      0         0       17h
replicaset.apps/prod-odoo-deployment-5dd6f98744  1       1         1       16h
replicaset.apps/prod-odoo-deployment-7d4dd4b55   0       0         0       17h
replicaset.apps/prod-odoo-deployment-95b65ddb5   0       0         0       16h

PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com\deploy\overlay\odoo-prod> kubectl get ingress -n odoo-prod
NAME          CLASS  HOSTS          ADDRESS          PORTS  AGE
prod-odoo-ingress  <none>  erp.decorhouse.tn  57.129.21.187  80, 443  47m
prod-pgadmin-ingress  <none>  pgadmin.odoo-dev.com  57.129.21.187  80          40m

PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com\deploy\overlay\odoo-prod> kubectl get certificates -n odoo-prod
NAME          READY  SECRET  AGE
odoo-prod  True    odoo-prod-secret  12h

PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com\deploy\overlay\odoo-prod>

```

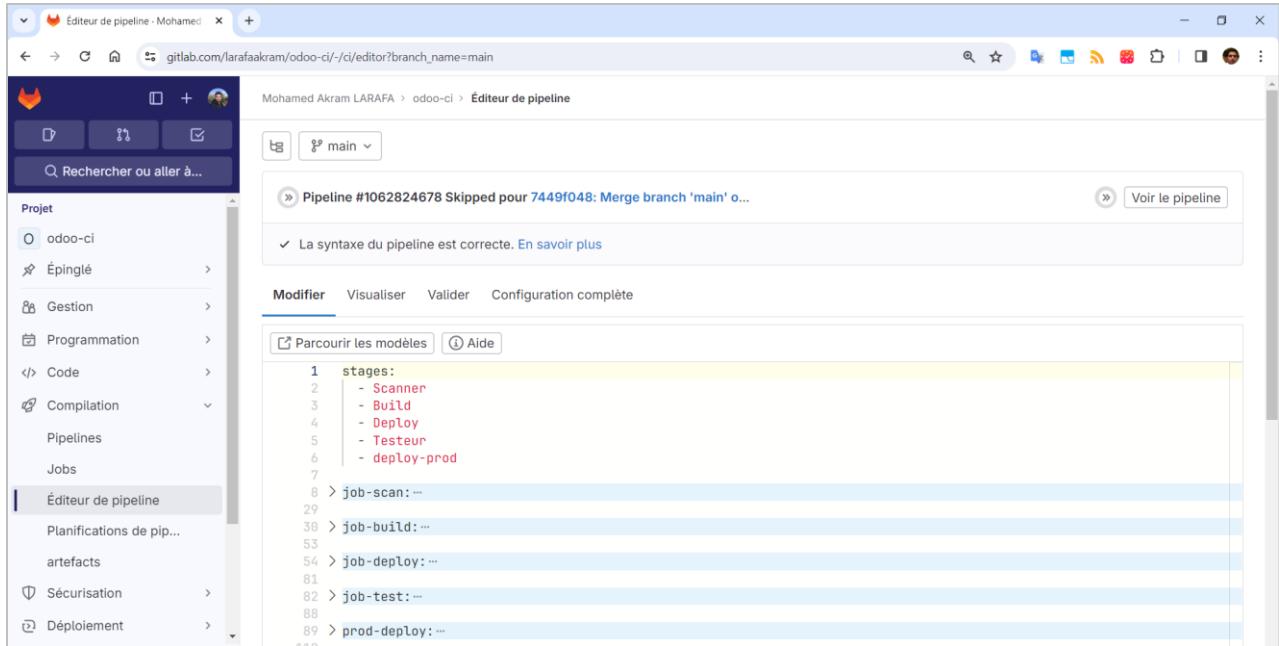
Figure 55 : Déploiement d'instance Odoo

CATÉGORIES					
Tous	Vente Des devis aux factures  ACTIVER	Facturation Factures & Paiements  ACTIVER	CRM Suivez vos pistes et signez des opportunités  ACTIVER		
Vente	EN SAVOIR PLUS	EN SAVOIR PLUS	EN SAVOIR PLUS		
Services					
Comptabilité					
Inventaire					
Fabrication					
Site web					
Marketing					
Ressources humaines					
Productivité					
Administration					
<b>MRP II</b>	Ordres de travail, Planification, Routes  EN SAVOIR PLUS	<b>Site web</b> Constructeur de sites web d'entreprise  ACTIVER	<b>Inventaire</b> Gérez votre stock et vos activités logistiques  ACTIVER		
<b>Comptabilité</b>	Comptabilité, Taxes, Budgets, Immobilisations...  EN SAVOIR PLUS	<b>Connaissances</b> Centralisez, gérez, partagez et développez votre bibliothèque de connaissances  EN SAVOIR PLUS	<b>Achats</b> Bons de commande, conventions et accords  ACTIVER		
<b>Point de vente</b>	Interface PdV conviaile pour les boutiques et les restaurants  ACTIVER	<b>Projet</b> Organisez et planifiez vos projets  ACTIVER	<b>eCommerce</b> Vendez vos produits en ligne  ACTIVER		
<b>Fabrication</b>	Ordres de fabrication & nomenclatures  ACTIVER	<b>Email Marketing</b> Concevez, envoyez et suivez des emails  ACTIVER	<b>Feuilles de temps</b> Suivez le temps & les coûts  EN SAVOIR PLUS		
<b>Notes de frais</b>	Soumettez, validez et facturez les notes de frais des employés  ACTIVER	<b>Studio</b> Créez et personnalisez des applications  ACTIVER	<b>Congés</b> Allouez des congés payés et suivez les demandes de congé  Mettre à niveau		

Figure 56 : Interface des applications Odoo

#### 5.4.2.10. Définition de notre script de pipeline « `.gitlab-ci.yaml` »

Notre pipeline est composé de cinq étapes (Scanner, Build, Deploy, Testeur, deploy-prod), comme le montre la figure 57.



The screenshot shows the GitLab CI Editor interface. On the left, there's a sidebar with project navigation. The main area displays a pipeline configuration file named `.gitlab-ci.yaml`. The code defines a single stage named `Scanner` which contains several steps: `Scanner`, `Build`, `Deploy`, `Testeur`, and `deploy-prod`. A message at the top indicates that the syntax is correct. Below the code editor, there are tabs for `Modifier`, `Visualiser`, `Valider`, and `Configuration complète`.

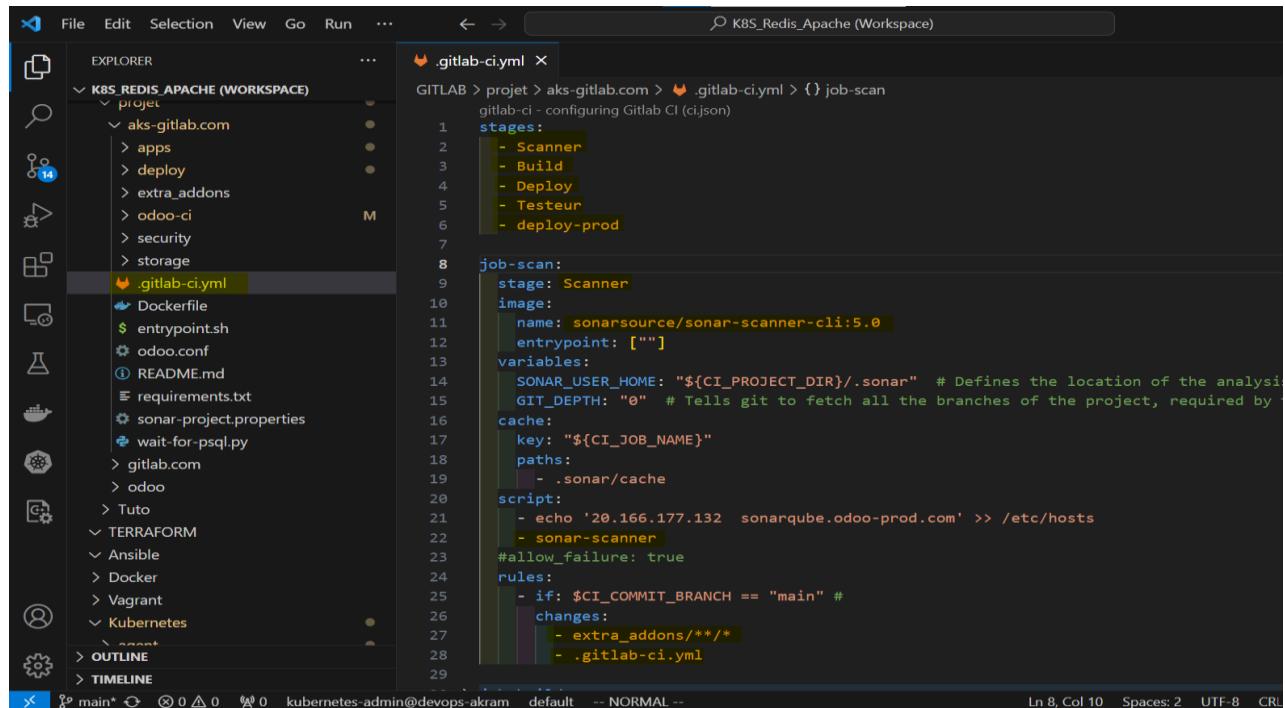
```

stages:
  - Scanner
  - Build
  - Deploy
  - Testeur
  - deploy-prod

```

Figure 57 : Liste des étapes du pipeline

Dans ce qui suit, nous pouvons également voir la définition des différents jobs d'étape dans les figures 58, 59, 60, 61.



The screenshot shows a code editor with the file `.gitlab-ci.yaml` open. The `Scanner` stage is highlighted in yellow. The code defines a job named `job-scan` with the following configuration:

```

stages:
  - Scanner
  - Build
  - Deploy
  - Testeur
  - deploy-prod

job-scan:
  stage: Scanner
  image:
    name: sonarsource/sonar-scanner-cli:5.0
  entrypoint: []
  variables:
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar" # Defines the location of the analysis
    GIT_DEPTH: "0" # Tells git to fetch all the branches of the project, required by cache
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - echo '20.166.177.132 sonarqube.odoo-prod.com' >> /etc/hosts
    - sonar-scanner
  allow_failure: true
  rules:
    - if: $CI_COMMIT_BRANCH == "main" #
      changes:
        - extra-addons/**/*
        - .gitlab-ci.yaml

```

Figure 58 : Définition de l'étape « Scanner » du pipeline

## Chapitre 5. Réalisation

The screenshot shows a GitLab CI pipeline configuration file (`.gitlab-ci.yml`) in a code editor. The file is located in a project named `K8S_REDIS_APACHE (WORKSPACE)` under the `aks-gitlab.com` namespace. The configuration defines a `job-build` stage with a `Build` job. This job uses a `gcr.io/kaniko-project/executor:debug` image, runs a `/kaniko/executor` script with context set to `CI_PROJECT_DIR`, and pushes to a registry with the image name `CI_REGISTRY_IMAGE` and commit SHA `CI_COMMIT_SHA`. It also skips TLS verification. The job depends on a previous `job-scan` and has rules for manual execution if the main branch is pushed. Changes made to the file include additions for extra addons and the `.gitlab-ci.yml` file itself.

```

GITLAB > projet > aks-gitlab.com > .gitlab-ci.yml > {} job-scan
3   - Build
4   - Deploy
5   - Testeur
6   - deploy-prod
7
8 > job-scan: ...
29
30 job-build:
31   stage: Build
32   image:
33     name: gcr.io/kaniko-project/executor:debug
34     entrypoint: []
35   tags:
36     - aks-runner
37   script:
38     - /kaniko/executor
39     --context "${CI_PROJECT_DIR}"
40     --dockerfile "${CI_PROJECT_DIR}/Dockerfile"
41     --destination "${CI_REGISTRY_IMAGE}:${CI_COMMIT_SHA}"
42     --skip-tls-verify
43   needs:
44     - job-scan
45   dependencies:
46     - "job-scan"
47   rules:
48     - if: $CI_COMMIT_BRANCH == "main"
49       when: manual
50   changes:
51     - extra-addons/**/*
52     - .gitlab-ci.yml

```

Figure 59 : Définition de l'étape « Build » du pipeline

The screenshot shows the continuation of the GitLab CI pipeline configuration. The `job-build` stage is followed by a `job-deploy` stage. This stage deploys to a Kubernetes cluster using a Bitnami `kubectl` image. It sets variables for the image name (`bitnami/kubectl:latest`), environment (`odoo-dev`), and stage (`Deploy`). It uses aks-runner tags and a specific image. A `before_script` block contains commands to set the cluster, credentials, context, and user. The `script` block applies the deployment overlay and sets the image to the odoo deployment. The `needs` section references the `job-build` stage. The `dependencies` section lists the `job-build` stage. Rules and changes sections are also present.

```

30 > job-build: ...
31
32 job-deploy:
33   variables:
34     IMAGE: "bitnami/kubectl:latest"
35     ENV: "odoo-dev"
36   stage: Deploy
37   tags:
38     - aks-runner
39   image:
40     name: $IMAGE
41     entrypoint: []
42   before_script:
43     - kubectl config set-cluster deploy-cluster --server="$K8S_SERVER" --insecure-skip-tls-verify
44     - kubectl config set-credentials gitlab --token=$(echo $K8S_TOKEN | base64 -d)
45     - kubectl config set-context deploy-cluster --cluster=deploy-cluster --namespace=$ENV --user=gitlab
46     - kubectl config use-context deploy-cluster
47   script:
48     -# kubectl apply -k deploy/overlay/$ENV
49     - kubectl set image deployment/dev-odoo-deployment odoo=${CI_REGISTRY_IMAGE}:$CI_COMMIT_SHA -n $ENV
50   needs:
51     - job-build
52   dependencies:
53     - "job-build"
54   rules:
55     - if: $CI_COMMIT_BRANCH == "main"
56       changes:
57         - extra-addons/**/*
58         - .gitlab-ci.yml

```

Figure 60 : Définition de l'étape « Deploy » du pipeline

```

GITLAB > projet > aks-gitlab.com > .gitlab-ci.yml > {} job-scan
30 > job-build: ...
53
54 > job-deploy: ...
81
82 > job-test: ...
88
prod-deploy:
variables:
  IMAGE: "bitnami/kubectl:latest"
  ENV: "odoo-prod"
stage: deploy-prod
tags:
  - aks-prod-runner
image:
  name: $IMAGE
  entrypoint: [""]
before_script:
  - kubectl config set-cluster prod-cluster --server="$K8S_PROD_SERVER" --insecure-skip-tls-verify
  - kubectl config set-credentials gitlab --token=$(echo $K8S_PROD_TOKEN | base64 -d)
  - kubectl config set-context prod-cluster --cluster=prod-cluster --namespace=$ENV --user=gitlab
  - kubectl config use-context prod-cluster
script:
  # kubectl apply -k deploy/overlay/$ENV
  - kubectl set image deployment/prod-odoo-deployment odoo=${CI_REGISTRY_IMAGE}:$CI_COMMIT_SHA -n $ENV
rules:
  - if: $CI_COMMIT_BRANCH == "production"
    changes:
      - extra-addons/**/*
      - .gitlab-ci.yml

```

Figure 61 : Définition de l'étape « deploy-prod » du pipeline

En termes de la section Configuration par un tableau illustre les adresses des portails déployés et configurés.

Services	Adresses
Portainer	<a href="http://portainer.odoo-dev.com/">http://portainer.odoo-dev.com/</a>
Sonarqube	<a href="http://sonarqube.odoo-dev.com">http://sonarqube.odoo-dev.com</a>
pgAdmin	<a href="http://pgadmin.odoo-dev.com:30520/">http://pgadmin.odoo-dev.com:30520/</a>
Instance Odoo	<a href="https://erp.decorhouse.tn/">https://erp.decorhouse.tn/</a>

Tableau 28 - Les portails utilisé et configuré

### 5.4.3. Supervision

La plateforme de surveillance nous fournit des informations cruciales qui nous aident à garantir la disponibilité du service et des performances optimales. Cela nous aide également à vérifier la disponibilité des nœuds du cluster AKS, en surveillant les ressources et les performances du système (CPU, espace disque, RAM, bande passante, etc.).

Pour déployer notre plateforme de surveillance, nous avons utilisé la commande ci-dessous :

```
# helm install prometheus-community/kube-prometheus-stack --create-namespace --namespace
prometheus --generate-name --set prometheus.service.type=LoadBalancer --set
prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=false --set
grafana.service.type=LoadBalancer --set grafana.adminpassword=M0nitoring##
```

## Chapitre 5. Réalisation

La figure 62 montre les ressources (CPU, RAM) utilisées par notre cluster de production :

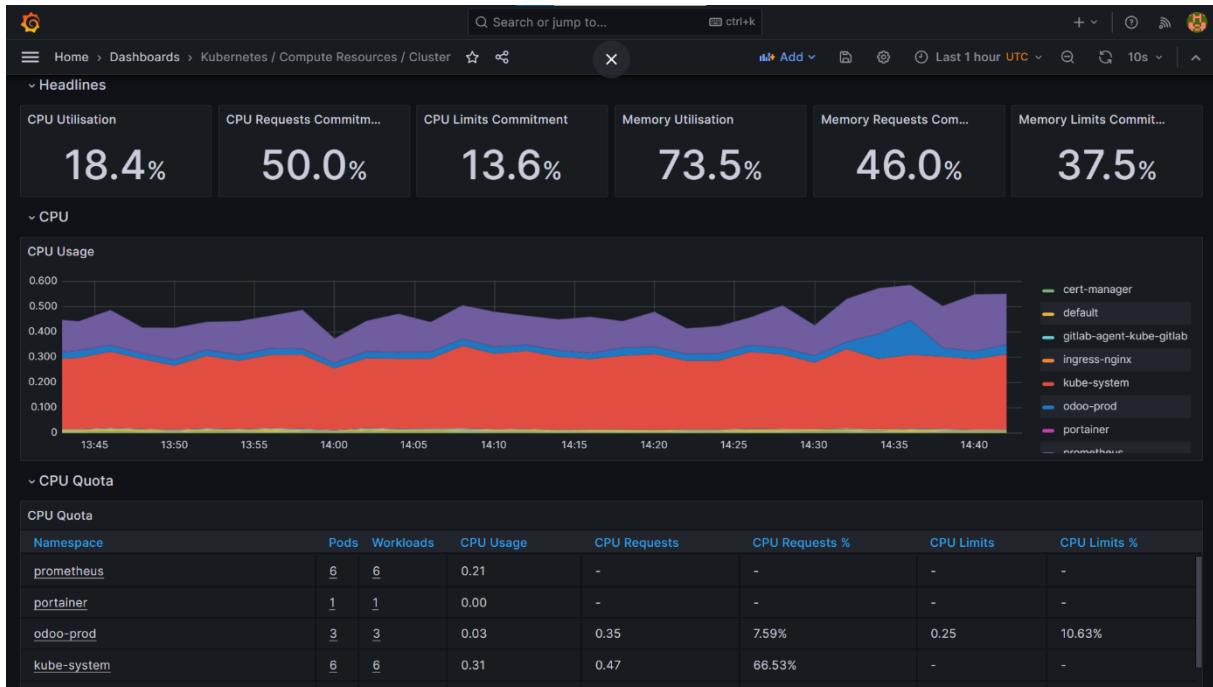


Figure 62 : Dashboard usage CPU et RAM dans le cluster de production

Les figures 63, 64 et 65 montrent l'usage du CPU, RAM et réseau de notre instance Odoo dans le cluster de production :

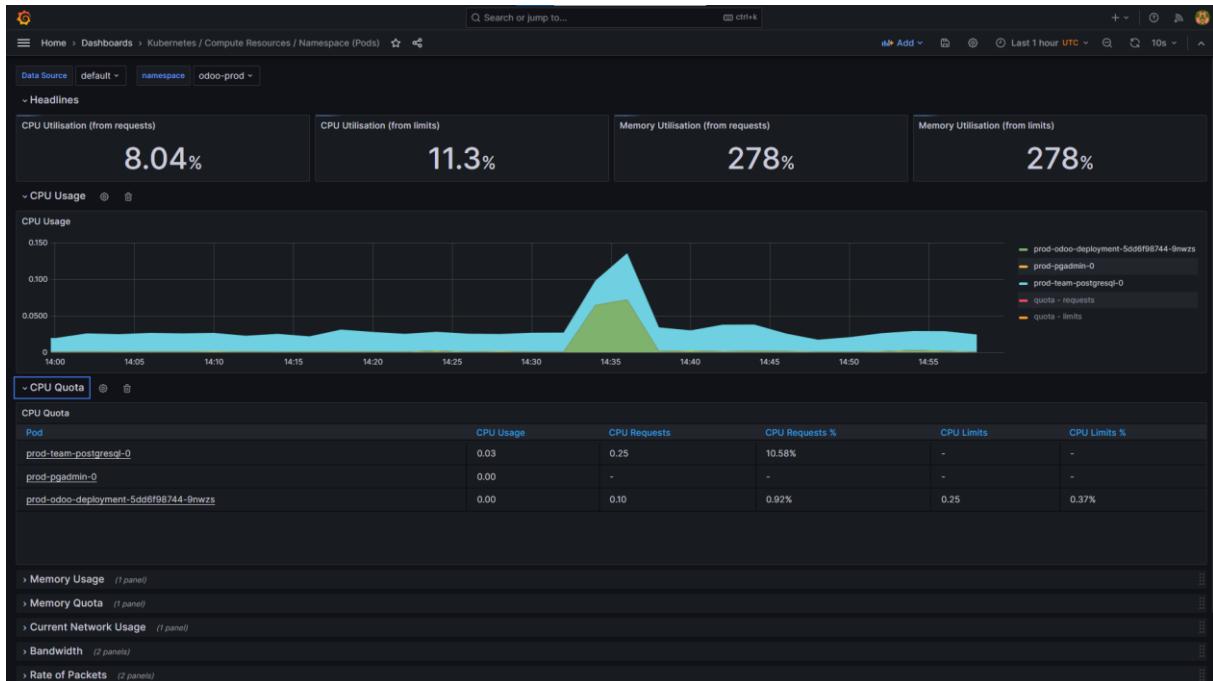


Figure 63 : Utilisation du CPU par l'instance Odoo en production

## Chapitre 5. Réalisation

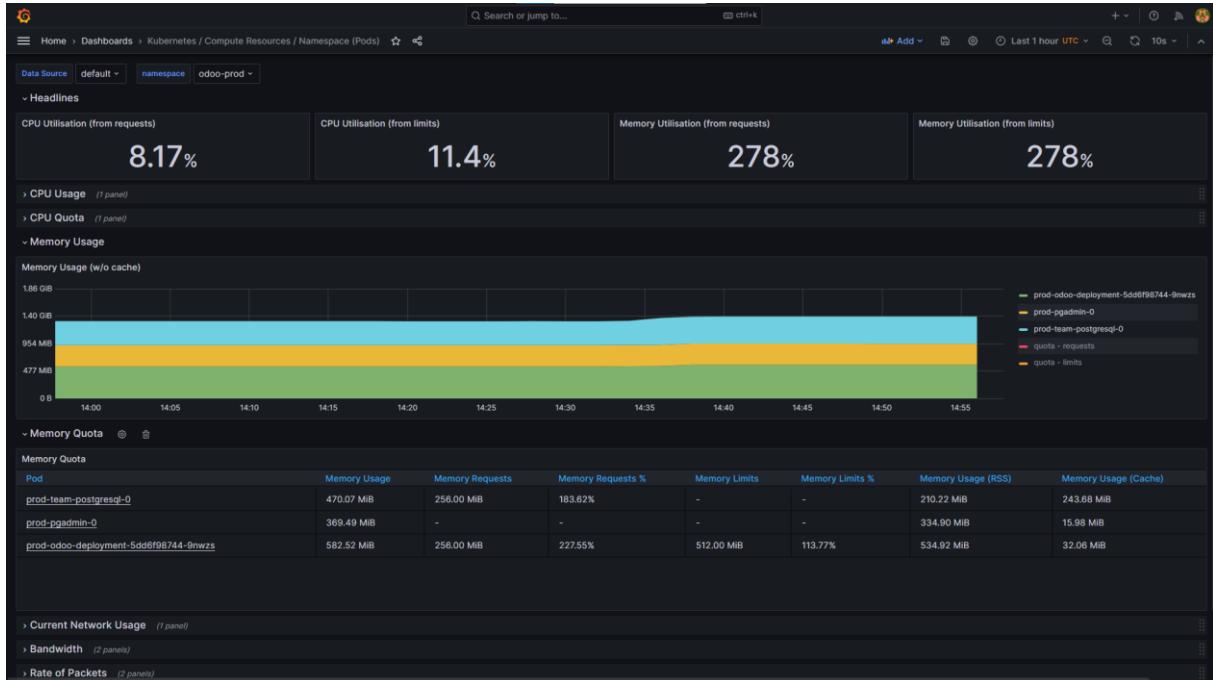


Figure 64 : Utilisation de RAM par l'instance Odoo en production

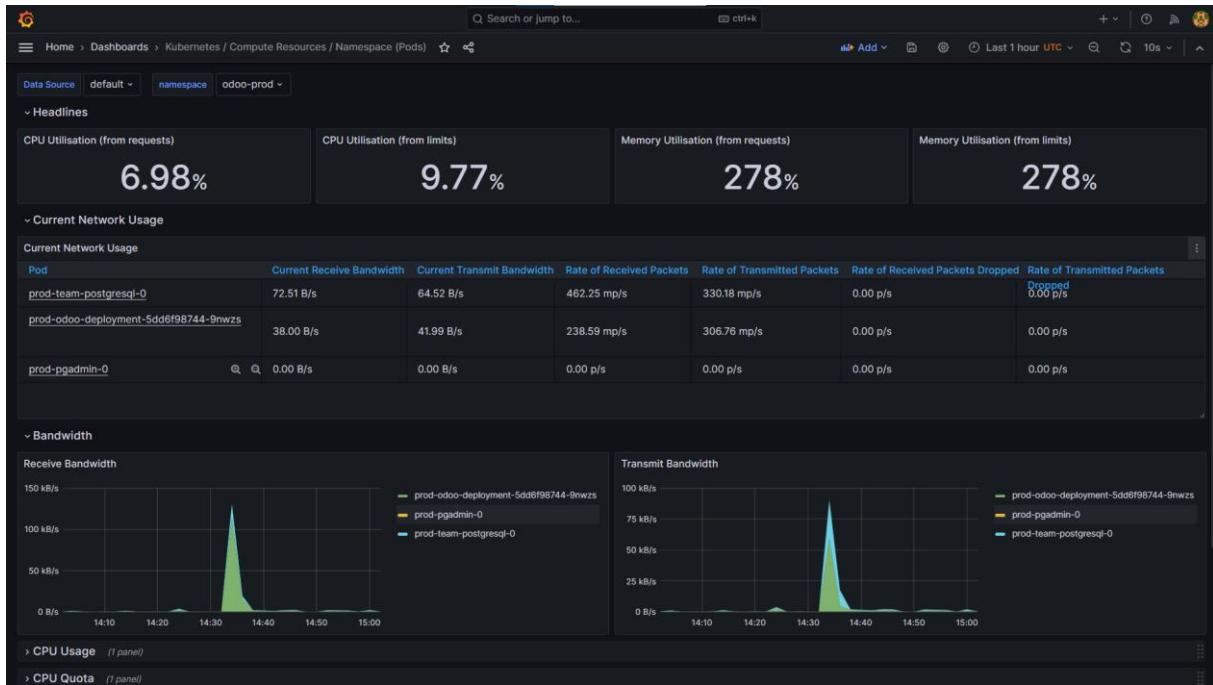


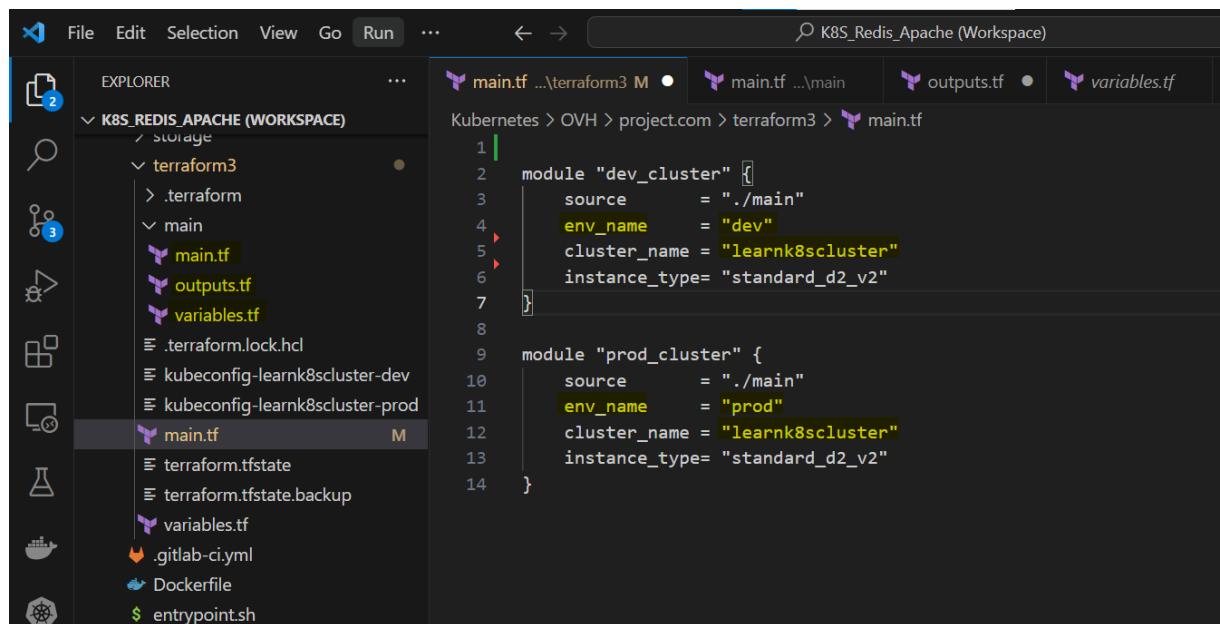
Figure 65 : Utilisation du réseau par l'instance Odoo en production

#### 5.4.4. Pré-configuration

##### 5.4.4.1 Crédit et provisionnement d'un cluster AKS

Afin de déployer et de provisionner nos deux clusters de production et de développement dans le nuage Azure, nous avons développé les modules suivants (figure 66) avec l'outil Terraform, qui définissent la source « ./main », le nom de l'environnement « env\_name », le nom du cluster « cluster\_name » et le type d'instance « instance\_type » pour chaque cluster.

*La source « ./main », qui contient la définition du fournisseur « azurerm », le groupe de ressources réservées à un cluster (name, location, dns\_prefix, default\_node\_pool {name, node\_count, vm\_size}) et les fichiers « outputs.tf » et « variables.tf », qui sont chargés de générer des « kubeconfig-% » (biais utilisés pour accéder aux clusters localement) et de déclarer des variables, respectivement.*



```

1 | module "dev_cluster" [
2 |   source      = "./main"
3 |   env_name    = "dev"
4 |   cluster_name = "learnk8scluster"
5 |   instance_type= "standard_d2_v2"
6 | ]
7 |
8 |
9 | module "prod_cluster" {
10 |   source      = "./main"
11 |   env_name    = "prod"
12 |   cluster_name = "learnk8scluster"
13 |   instance_type= "standard_d2_v2"
14 | }

```

Figure 66 : Définition des clusters avec des modules Terraform

## Chapitre 5. Réalisation

```

Kubernetes > OVH > project.com > terraform3 > main > outputs.tf
1
2
3 resource "local_file" "kubeconfig" [
4   depends_on = [azurerm_kubernetes_cluster.cluster]
5   filename   = format("kubeconfig-%s", azurerm_kubernetes_cluster.cluster.name)
6   content    = azurerm_kubernetes_cluster.cluster.kube_config_raw
7 ]
  
```

Figure 67 : Définition de la création de kubeconfig à l'aide de local\_file

L'exécution de ce code est effectuée en deux étapes :

```
# terraform plan // la création d'un plan, qui permet de simuler et de voir les ressources à créer
# terraform apply // appliquer le plan
```

Les figures 68 et 69 montrent le résultat de l'exécution pour la création de nos deux clusters AKS avec leurs éléments :

Paramètre	Valeur
Groupe de ressources	learnk8sResourceGroup-dev
Status	Réussi (en cours d'exécution)
Emplacement	North Europe
Abonnement	Azure subscription 1
ID d'abonnement	b4709a3d-f7a3-43ab-aa93-03f858c443b8
Étiquettes (modifiées)	: Ajouter des étiquettes

Service	Détails
Services Kubernetes	Type de chiffrement : Chiffrement au repos avec une clé gérée par la plateforme Pools de nœuds virtuels : Non activé
Pools de nœuds	Pools de nœuds : 1 pool de nœuds Versions de Kubernetes : 1.26.6 Tailles de nœud : standard_d2_v2
Configuration	Version de Kubernetes : 1.26.6 Type de mise à niveau automatique : Désactivé Authentification et autorisation : Comptes locaux avec RBAC Kubernetes
Mise en réseau	Adresse du serveur d'API : oamk8scluster-b5becc22.hcp.northeurope.azureaks.io Type de réseau (plug-in) : Kubenet CIDR de pod : 10.244.0.0/16 CIDR de service : 10.0.0.0/16 CIDR du pont Docker : - Stratégie réseau : Aucun Équilibrage de charge : Standard Routeur d'application HTTP : Activé Cluster privé : Non activé Plages d'adresses IP autorisées : Non activé Contrôleur d'entrée de la : Non activé

Figure 68 : Cluster AKS de l'environnement de développement

## Chapitre 5. Réalisation

The screenshot shows the Azure portal interface for managing an AKS cluster. The left sidebar lists various Azure services. The main content area is focused on the 'learnk8scluster-prod' cluster. The 'Properties' tab is active, showing detailed configuration for the cluster, including its group, status (Success), location (North Europe), and specific Kubernetes settings like version (1.26.6) and node pool (1 pool de nœuds). The right side of the screen displays the 'Network' configuration section.

Figure 69 : Cluster AKS de l'environnement de production

La figure 70 présente une liste des objets qui composent les éléments nécessaires au bon fonctionnement de nos clusters d'AKS.

This screenshot shows a list of resources created for the AKS clusters. The table includes columns for Name, Type, Location, Group of resources, Subscription, and Last modified. Key resources listed include the Azure subscription, the AKS clusters themselves, service and storage accounts, and various network and security components.

Nom	Type	Emplacement	Groupe de ressources	Abonnement	Dernier
Azure subscription 1	Abonnement			Azure subscription 1	il y a 2
learnk8scluster-prod	service Kubernetes	North Europe	learnk8sResourceGroup-prod	Azure subscription 1	il y a 2
learnk8scluster-dev	service Kubernetes	North Europe	learnk8sResourceGroup-dev	Azure subscription 1	il y a 2
kubernetes	Équilibrage de charge	North Europe	mc_Jeank8sresourcegroup-dev_jeank8s...	Azure subscription 1	il y a 1
learnk8sResourceGroup-prod	Groupe de ressources	North Europe	learnk8sResourceGroup-prod	Azure subscription 1	il y a 4
kubernetes	Équilibrage de charge	North Europe	MC_Jeank8sResourceGroup-prod_jeank...	Azure subscription 1	il y a 4
67c55398-f954-4b16-897c-636c02cee978	Adresse IP publique	North Europe	MC_Jeank8sResourceGroup-prod_jeank...	Azure subscription 1	il y a 7
MC_Jeank8sResourceGroup-prod_jeank8scluster-prod_northeurope	Groupe de ressources	North Europe	MC_Jeank8sResourceGroup-prod_jeank...	Azure subscription 1	il y a 7
aks-vnet-22454606	Réseau virtuel	North Europe	MC_Jeank8sResourceGroup-prod_jeank...	Azure subscription 1	il y a 7
learnk8sResourceGroup-dev	Groupe de ressources	North Europe	learnk8sResourceGroup-dev	Azure subscription 1	il y a 7
4abcc6947-9e23-4744-a60b-7d1240b172ed	Adresse IP publique	North Europe	MC_Jeank8sResourceGroup-dev_jeank8...	Azure subscription 1	il y a 8
13f34e669a4c43ae8e9a.northeurope.aksapp.io	Zone DNS	Global	MC_Jeank8sResourceGroup-dev_jeank8...	Azure subscription 1	il y a 11

Figure 70 : Liste des ressources créées pour chaque cluster AKS

### 5.4.4.2 Crédit et configuration d'un compte gitlab.com

La figure 71 montre l'identification du compte que j'ai créé sur la plateforme Gitlab.

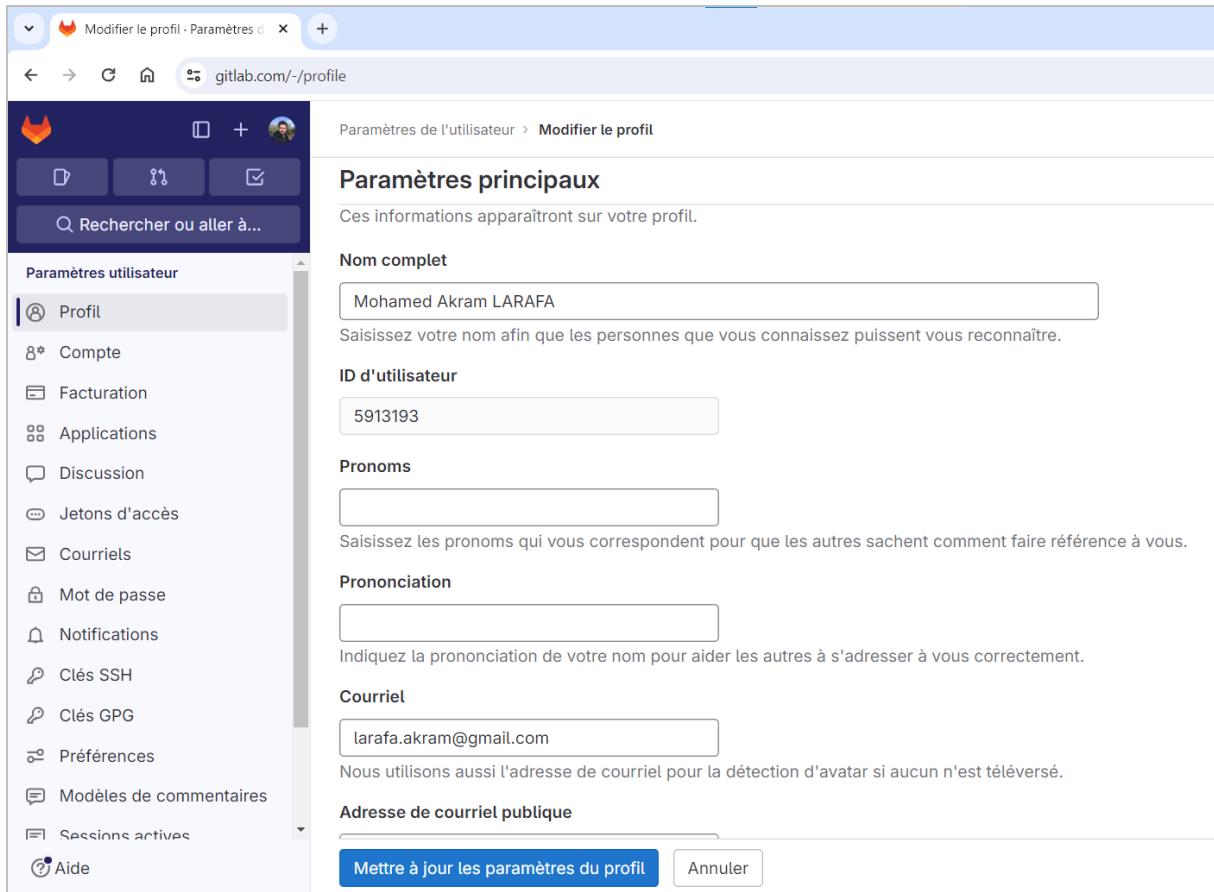


Figure 71 : Identification du compte Gitlab

Après avoir créé un compte sur la plateforme Gitlab, nous passons à la phase de préparation, qui comprend les points suivants :

- Ajouter un cluster Kubernetes
- Ajouter un Runner
- Intégrer SonarQube

### Ajouter un cluster Kubernetes

1. Naviguer vers votre projet, cliquez sur « Opération » → « Clusters Kubernetes »
2. Cliquer sur le bouton « Connecter un cluster », créer l'agent puis « Enregistrer »

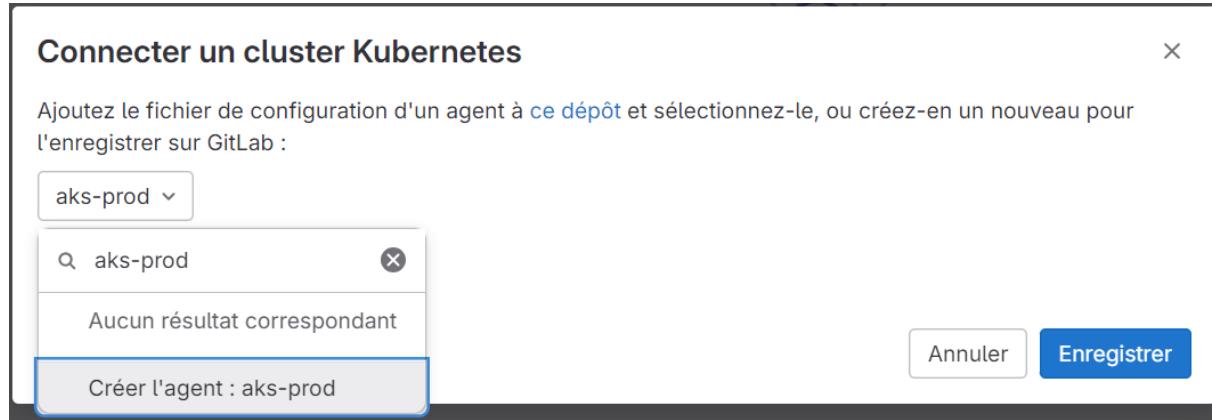


Figure 72 : Création d'un agent au niveau Gitlab

3. La figure 73 montre une boîte de dialogue qui s'affiche et donne un jeton d'accès avec des commandes HELM pour installer l'agent sur l'AKS.

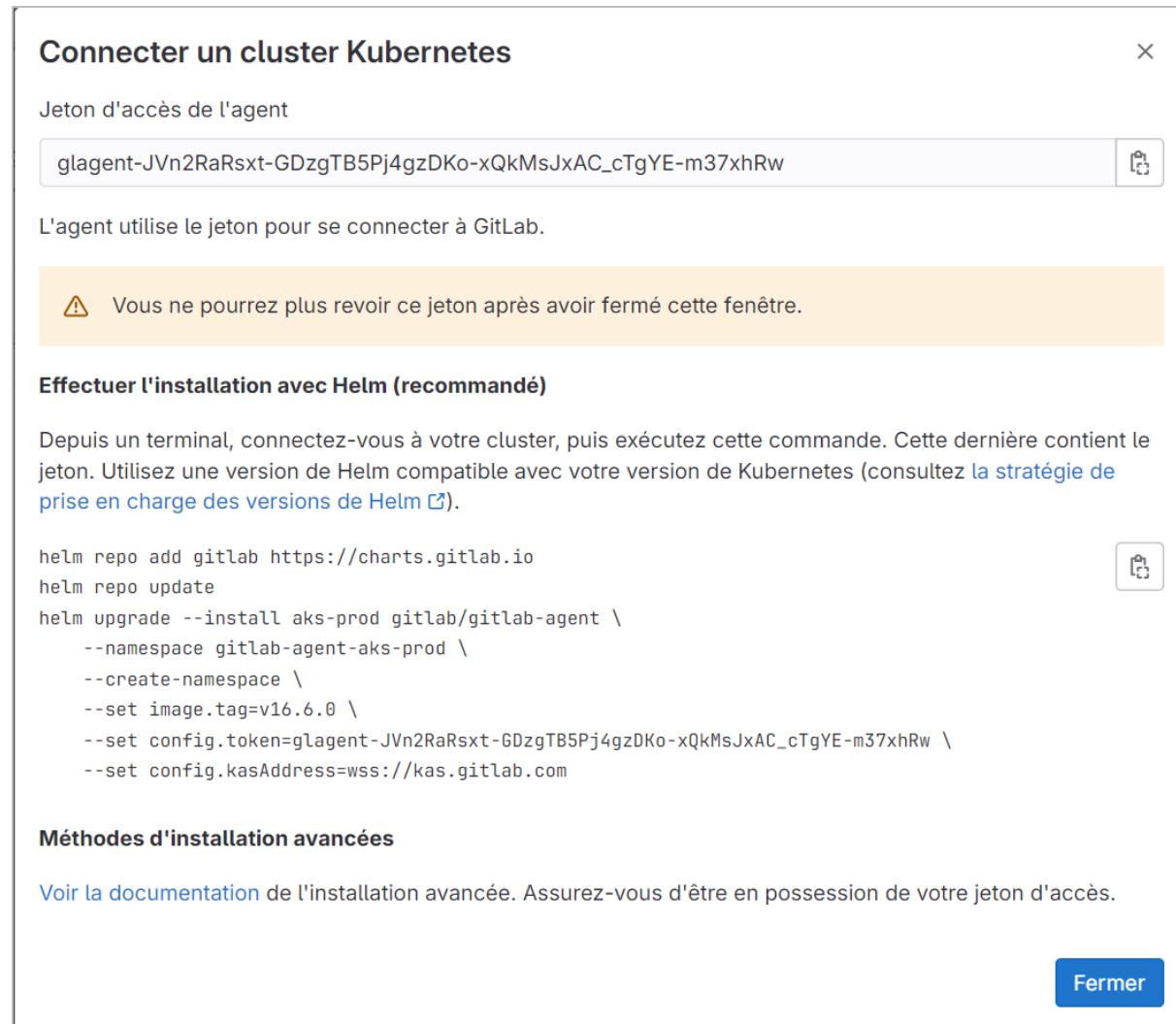
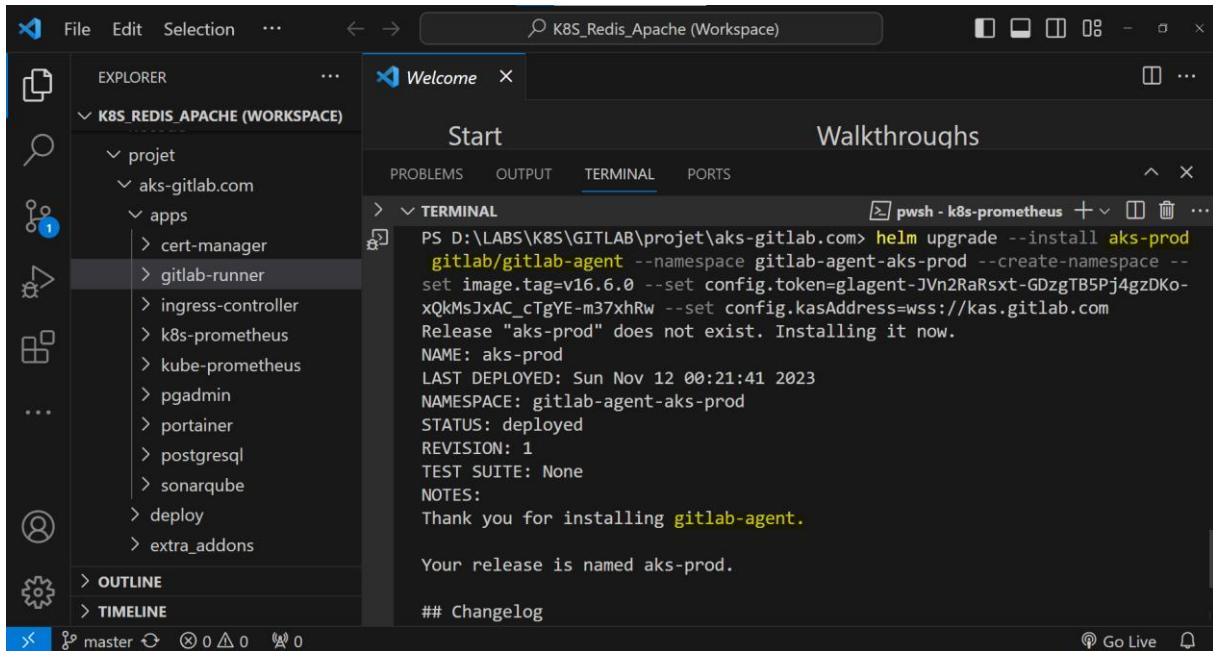


Figure 73 : Les instructions d'installation d'agent Gitlab sur l'AKS

4. La figure 74 montre l'installation de l'agent sur le cluster AKS.



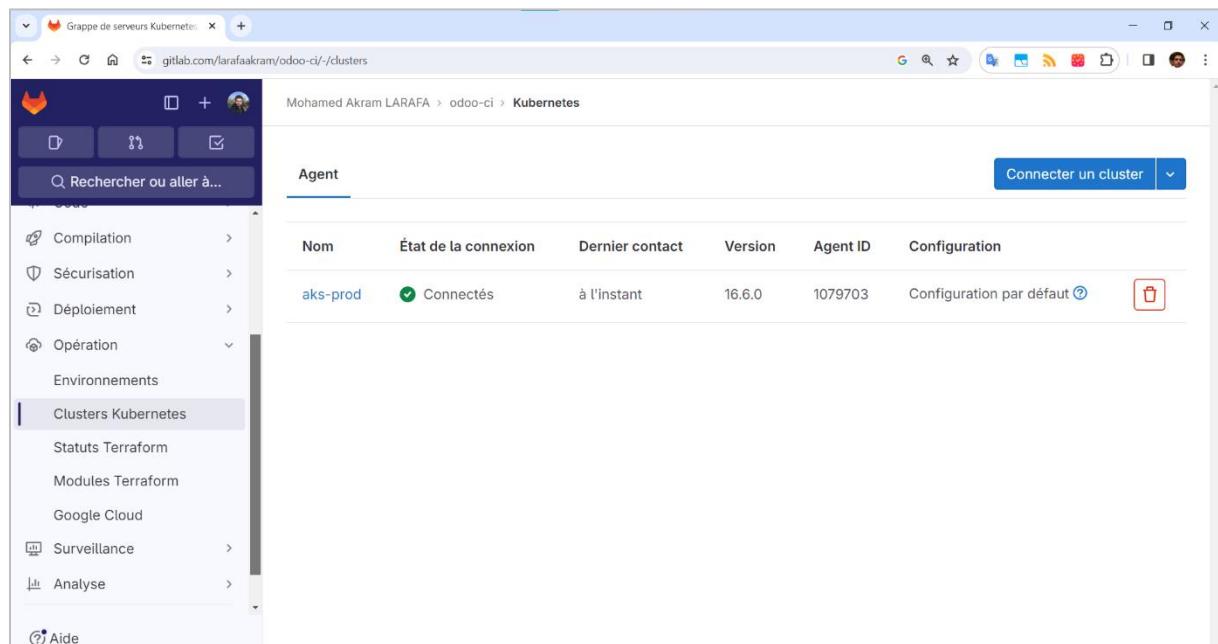
```
PS D:\LABS\K8S\GITLAB\projet\aks-gitlab.com> helm upgrade --install aks-prod
gitlab/gitlab-agent --namespace gitlab-agent-aks-prod --create-namespace --
set image.tag=v16.6.0 --set config.token=glagent-JVn2RaRsxt-GDzgTB5Pj4gzDKo-
xQkMsJxAC_CtgYE-m37xhRw --set config.kasAddress=wss://kas.gitlab.com
Release "aks-prod" does not exist. Installing it now.
NAME: aks-prod
LAST DEPLOYED: Sun Nov 12 00:21:41 2023
NAMESPACE: gitlab-agent-aks-prod
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing gitlab-agent.

Your release is named aks-prod.

## Changelog
```

Figure 74 : Installation de l'agent sur le cluster AKS

5. La figure 75 montre que la plateforme Gitlab est connectée à l'AKS via son agent.



Nom	État de la connexion	Dernier contact	Version	Agent ID	Configuration
aks-prod	Connectés	à l'instant	16.6.0	1079703	Configuration par défaut

Figure 75 : Gitlab plateforme connecté à l'AKS

### Ajouter un Runner

1. Naviguer vers votre Projet, Paramètres → CI/CD, puis cliquez « Runners »
2. Cliquez sur « Nouvel runner du projet », nos paramètres étaient (Système d'exploitation : linux, Étiquette : k8s-runner), puis valider la création.

3. En utilisant le jeton généré après la validation et l'étiquette défini dans la configuration « values.yaml » de notre chart HELM.

```
# helm install --namespace default gitlab-runner-k8s -f .\values.yaml gitlab/gitlab-runner
```

4. La Figure 76 montre que le Runner ajouté à notre projet est prêt à être utilisé pour exécuter les jobs de pipeline.

The screenshot shows the GitLab interface for managing CI/CD runners. On the left, there's a sidebar with project navigation (odoo-ci, Tickets, Requests, Repository, Pipelines) and a 'Paramètres CI/CD' section which is currently selected. The main content area is titled 'Paramètres CI/CD' and shows information about runners. It defines two states: 'actif' (available to run jobs) and 'en pause' (not available). A note explains that labels control which job types a runner can execute. Below this, a section titled 'Runners de projet' lists runners assigned to the project, with a button to add a new runner. Under 'Runners de projet assignés', two runners are listed: '#29447789 (wzEz5qvGG)' and 'k8s-runner'. Each runner has edit and delete icons next to it.

Figure 76 : Le Runner est assigné à notre projet Gitlab

## Intégrer SonarQube

1. Naviguer vers votre projet, Paramètres → Jetons d'accès → Ajouter un nouveau jeton
2. Définir le nom, le rôle comme « Reporter » et le niveau de permission à « api » puis valider, un jeton d'accès personnel sera généré pour l'étape suivantes au niveau Sonarqube.

3. Sonarqube : Accédez à Administration → Configuration → Paramètres généraux → Intégrations de la plateforme DevOps → sélectionnez l'onglet Gitlab → spécifiez les paramètres (nom de configuration, l'URL de l'API Gitlab, jeton d'accès personnel) puis valider.

Figure 77 : Connexion établie entre Sonarqube et Gitlab

4. Sonarqube : Projets → Nouveau projet → Choisir l'option « From Gitlab » (Manuelle) → Créer projet (Valider) → Choisir l'option (with Gitlab CI)
  - a. Généré un jeton (sera utilisé dans la définition des variables coté Gitlab)
  - b. Sonarqube URL (aussi sera utilisé dans la définition des variables cotés Gitlab)
  - c. Sélection de l'environnement, python dans notre cas (les paramètres fourni sera ajouté au fichier « sonar-project.properties » coté Gitlab).
  - d. Copier le bloc du code « sonarqube-check », ajouter le comme job dans fichier « .gitlab-ci.yml » coté Gitlab.
5. Gitlab : Naviguer vers votre Projet, Paramètres → CI/CD → Variables

Créer les variables :

- SONAR\_HOST\_URL
- SONAR\_TOKEN
- SONAR\_LOGIN
- SONAR\_PASSWORD

Les variables ci-dessus seront utilisées par le job de pipeline « sonarqube-check » pour se connecter au serveur Sonarqube.

### 5.5. Conclusion

Ce chapitre nous aide à présenter l'environnement de travail qui définira les différents modèles sur lesquels notre solution est basée, puis introduit les technologies et les outils que nous avons utilisés pour réaliser ce travail. Enfin, nous présentons des captures d'écran illustrant les résultats de l'exécution des différents processus.

## Conclusion Générale

Ce travail s'inscrit dans le cadre d'un mémoire de Master et a été réalisé au sein de CIAM Solution. Il porte sur la création d'une chaîne d'intégration et de déploiement continu basée sur Kubernetes, accompagnée d'une table de monitoring. Nous avons divisé notre travail en cinq modules : un module d'intégration continue, un module de déploiement continu, un module de configuration, un module de surveillance et un module de pré-configuration.

Pour compléter ce travail, nous présentons d'abord l'idée générale de notre stage, qui consiste à présenter l'entreprise hôte, l'étude des constats, l'étude de l'existant et constatations, objectifs et contributions, et enfin en appliquant la méthodologie choisie. Nous procédons ensuite à des recherches préliminaires, en définissant le cloud computing et ses différents modèles, puis en définissant les pratiques utilisées dans notre chaîne Devops, puis en présentant les objets et notions liés à l'architecture et à la plateforme utilisées, puis comparant les différentes solutions techniques qui s'offrent à nous et résumant enfin nos choix techniques. Nous nous intéressons ensuite à l'analyse et à la spécification, et ce chapitre se concentre sur l'analyse de l'identification des acteurs, de leurs besoins et de leurs représentations. Nous poursuivons avec l'architecture et la conception détaillées, ce qui nous permet de démontrer l'architecture globale de la solution et d'analyser ses aspects dynamiques. Enfin, nous avons entamé la phase réalisation, qui présente notre environnement de travail (qu'il s'agisse d'infrastructure, plateformes et outils), la présentation des technologies que nous avons utilisées pour effectuer notre travail, et la finalisation de la présentation de quelques interfaces liées aux résultats de l'exécution des tâches du pipeline, au déploiement manuel, à l'affichage des mesures ou à la configuration et à l'approvisionnement de l'infrastructure.

Cette expérience nous a permis de découvrir de nouvelles approches et plateformes telles que Gitlab, kubernetes et docker, intégrer avec succès de nouveaux systèmes et logiciels dans et avec différents modèles de cloud computing, tels que le déploiement d'un cluster dans le cloud Azure à l'aide de Terraform.

D'un point de vue personnel, le projet a également été bénéfique, car il nous a donné un aperçu du travail dans une grande entreprise et à un niveau professionnel. Travailler en équipe, gérer des projets de manière méthodique et partager la charge de travail sont autant de qualités que j'ai apprises grâce à cette mémoire.

Un certain nombre d'améliorations intéressantes apportées à notre projet méritent d'être mentionnées ici. Nous pensons qu'il est souhaitable d'ajouter des tests unitaires au niveau du

## Conclusion générale

pipeline d'intégration continu pour vérifier que le code renvoie les bons résultats. Ils permettent à l'équipe de développement de logiciels d'éprouver tout le code de la base, d'isoler les erreurs, de trouver les défauts cachés et de réduire le temps de recherche des défauts. Dans notre cas, les développeurs doivent utiliser le framework de test unitaire original de Python, PyUnit.

Nous pensons également que l'ajout de l'outil Trivy pour l'analyse des vulnérabilités de nos images docker créées dans la phase d'intégration continue du pipeline peut nous donner l'assurance que tout va bien avec nos instances Odoo. Cela nous donne la possibilité de trouver des vulnérabilités dans les images de conteneurs et de les corriger avant de pousser l'image dans un registre ou de l'exécuter en tant que conteneur.

Nous espérons enfin que ce travail satisferont les dirigeants de la société d'accueil « CIAM Solution » et les membres du jury.

## Netographie

- [1] tuleap, «Comprendre la méthode Agile Scrum en 10 min,» tuleap, [En ligne]. Available: <https://www.tuleap.org/fr/agile/comprendre-methode-agile-scrum-10-minutes>. [Accès le 25 07 2023].
- [2] F. MILLOGO, «Mise en place d'une application webmapping de géolocalisation des points d'intérêt de la ville de Ouagadougou,» MEMOIRE Online, 2012. [En ligne]. Available: [https://www.memoireonline.com/05/13/7195/m\\_Mise-en-place-dune-application-webmapping-de-geolocalisation-des-points-dintert-de-la-vill6.html](https://www.memoireonline.com/05/13/7195/m_Mise-en-place-dune-application-webmapping-de-geolocalisation-des-points-dintert-de-la-vill6.html). [Accès le 20 07 2023].
- [3] cloudbees, «What is Continuous Integration: Testing, Software & Process Tutorial,» cloudbees, 2023. [En ligne]. Available: <https://www.cloudbees.com/continuous-delivery/continuous-integration>. [Accès le 20 10 2023].
- [4] S. PITTEL, «Continuous integration vs. delivery vs. deployment,» atlassian, [En ligne]. Available: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment#:~:text=the%20main%20branch.-,Continuous%20delivery,-Continuous%20delivery%C2%A0is>. [Accès le 20 08 2023].
- [5] S. PITTEL, «Continuous integration vs. delivery vs. deployment,» atlassian, [En ligne]. Available: <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment#:~:text=Read%20more-,Continuous%20deployment,-Continuous%20deployment%C2%A0goes>. [Accès le 20 08 2023].
- [6] Redhat, «Qu'est-ce que Kubernetes (k8s)?,» Redhat, 27 03 2020. [En ligne]. Available: <https://www.redhat.com/fr/topics/containers/what-is-kubernetes>. [Accès le 20 07 2023].
- [7] Redhat, «Qu'est-ce qu'un pod Kubernetes ?,» Redhat, 31 10 2017. [En ligne]. Available: <https://www.redhat.com/fr/topics/containers/what-is-kubernetes-pod>. [Accès le 27 07 2023].

- [8] datascientest, «Docker : qu'est-ce que c'est et comment l'utiliser ?,» datascientest, 23 Décembre 2022. [En ligne]. Available: <https://datascientest.com/docker-guide-complet>. [Accès le 20 07 2023].
- [9] Ajdaini-Hatim, «Gérer vos déploiements kubernetes (k8s) avec Kustomize,» 06 Juillet 2021. [En ligne]. Available: <https://devopssec.fr/article/deploiements-k8s-avec-kustomize>.
- [10] T. K. Authors, «Ingress | Kubernetes,» [En ligne]. Available: <https://kubernetes.io/fr/docs/concepts/services-networking/ingress/>. [Accès le 20 Juin 2023].
- [11] linux-console.net, «Qu'est-ce que Cert-Manager et comment configurer Cert-Manager pour les certificats SSL dans le cluster Kubernetes sur AWS à l'aide de Helm,» linux-console.net, [En ligne]. Available: <https://fr.linux-console.net/?p=3037>. [Accès le 20 05 2023].
- [12] I. Amazon Web Services, «Quelle est la différence entre Docker et une machine virtuelle ?,» [En ligne]. Available: <https://aws.amazon.com/fr/compare/the-difference-between-docker-vm/>. [Accès le 15 Août 2023].
- [13] S. Kumar, «Terraform vs. Ansible: Key Differences and Comparison of Tools,» k21academy, 5 09 2023. [En ligne]. Available: <https://k21academy.com/ansible/terraform-vs-ansible/>.
- [14] ambient-it, «TOUT SAVOIR SUR LES FICHIERS YAML EN 5 MINUTES,» ambient-it, 10 03 2023. [En ligne]. Available: <https://www.ambient-it.net/tout-savoir-fichiers-yaml/>. [Accès le 12 09 2023].
- [15] Cegle, «ODOO: logiciel de gestion,» Cegle, [En ligne]. Available: [https://www.celge.fr/editeurs/odoo-logiciel-de-gestion-dentreprise-crm-erp-facturation-comptabilite-gpao-cms-e-commerce?autocomplete\\_\\_contacter\\_1\\_editeur=Odoo](https://www.celge.fr/editeurs/odoo-logiciel-de-gestion-dentreprise-crm-erp-facturation-comptabilite-gpao-cms-e-commerce?autocomplete__contacter_1_editeur=Odoo). [Accès le 20 09 2023].
- [16] Oracle, «Qu'est-ce que PostgreSQL? | Oracle France,» Oracle, 2023. [En ligne]. Available: <https://www.oracle.com/fr/database/definition-postgresql/>. [Accès le 20 09 2023].

- [17] cubic, «Télécharger pgAdmin 4 (gratuit) Windows, Linux, Mac - Clubic,» cubic, 2023. [En ligne]. Available: <https://www.clubic.com/telecharger-fiche322084-pgadmin.html>. [Accès le 25 08 2023].
- [18] S. WIKI, «Qu'est-ce que Kaniko ?,» 5 Avril 2022. [En ligne]. Available: <https://wiki.sfeir.com/kubernetes/outils/kaniko/>.
- [19] G. Booch, «Intégration continue,» wikipédia, 19 01 2023. [En ligne]. Available: [https://fr.wikipedia.org/wiki/Int%C3%A9gration\\_continue](https://fr.wikipedia.org/wiki/Int%C3%A9gration_continue). [Accès le 20 07 2023].

## ملخص

لتحكم الجيد وتحسين ادارت التطبيقات، منذ تحميل الكود نحو الخادم المسؤول على الاصادرات إلى ان يتم تسليم التطبيق وجميع اجزائه الى الحريف، دأبت شركة سيام سوليسيون إلى وضع سلسلة ادماج ونشر كود مستمر من اجل برمجة هذه العملية، مع استخدام اطر ونماذج مختلفة للحوسبة السحابية، ويمكن الاشراف عليها من خلال لوحة متابعة. في هذا السياق، كمشروع ماجستير، قمنا بتنفيذ الحل باستخدام العديد من التكنولوجيات مثل:

Kaniko

الكلمات المفتاحية: الحوسبة السحابية، التوفير العالي، الخدمة المدارة، التكامل المستمر، النشر المستمر.

## Résumé

Afin d'améliorer la gestion du cycle de vie de l'application depuis la soumission du code source aux serveurs de gestion de versions jusqu'à la livraison de la release, Ciam Solution a choisi de construire un pipeline d'intégration et de déploiement continu pour automatiser le processus ci-dessus avec l'utilisation de différents Framework et modèle du cloud computing. Tout ceci doit être surveillé et affiché dans le tableau de bord de surveillance. Dans ce contexte, en tant que projet de mastère, nous avons implémenté la solution en utilisant plusieurs technologies telles que Portainer, Kubernetes, Docker, Terraform, Gitlab, Sonarqube et Kaniko.

**Mots clés :** cloud computing, haute disponibilité, service managé, intégration continue, déploiement continu.

## Abstract

In order to improve application lifecycle management, from source code submission to release management servers to release delivery, Ciam Solution decided to build a continuous integration and deployment pipeline to automate the above process, using different frameworks and cloud computing models. All of this would be monitored and displayed in the monitoring dashboard. In this context, as a master's project, we implemented the solution using various technologies such as Portainer, Kubernetes, Docker, Terraform, Gitlab, Sonarqube and Kaniko.

**Keywords:** cloud computing, high availability, managed service, continuous integration, continuous deployment.