

Dédicace

“

*J'e tiens à dédier ce modeste travail aux personnes qui me
sont les plus chers et tout particulièrement :,*

*Mes chers parents Pour leur affection et leur appui qu'ils
n'ont cessé de me donner durant tout mon travail, À mon
cher frère ,À mes chers soeurs ,*

*À mes amis qui n'ont épargné aucun effort pour me
soutenir tout au long de mon cursus d'études. Et à tous
ceux qui m'ont soutenu durant ce projet de fin d'études.*

Que Dieu vous bénisse ,

À tous ceux qui me sont chers, à vous tous

Merci.

”

- safwen ben fredj

Remerciements

Je tiens à remercier tous ceux qui ont contribué à la réalisation de ce travail et en premier lieu, mon encadrant Monsieur pour son aide, sa disponibilité, et ses précieux conseils. Ma gratitude s'adresse, aussi, au jury pour m'avoir fait le grand honneur d'accepter d'évaluer ce travail. Qu'ils trouvent ici l'expression de mon profond respect.

Mes remerciements s'adressent également aux membres de société "roundesk" qui m'ont offert l'ambiance favorable pour le déroulement de mon projet et je tiens à remercier spécialement Madame **Fekih Ibtihel** mon encadrante qui m'a beaucoup aidé et orienté qu'il trouve ici le témoignage de ma profonde reconnaissance. Je tiens également à exprimer ma gratitude à mes formateurs et mes amis à l'U.V.T, pour leurs aides pour mener à terme ce travail.

Finalement, je tiens à remercier tous ceux qui, de près ou de loin, m'ont apporté un soutien moral ou matériel.

Table des matières

Dédicace	I
Remerciements	II
1 Introduction générale	1
Introduction générale	1
2 Cadre général du projet	3
2.1 Introduction	4
2.2 Présentation de Roundesk Technologies	4
2.3 Objectifs	4
2.4 Analyse de l'existant	5
2.5 Problématique	6
2.6 Solution proposée	6
2.6.1 Conclusion	7
3 Intégration continue/Livraison continue, Déploiement continu	8
3.1 Introduction	9
3.2 Approche intégration, livraison déploiement continus	9
3.2.1 Intégration continue (CI)	10
3.2.2 La livraison continue/déploiement continu (CD)	10
3.3 Définition de DevOps	11
3.4 Approche virtualisation / conteneurisation	11
3.4.1 La virtualisation	12
3.4.2 La Conteneurisation	12
3.5 Conclusion	13

4	Architecture et principe de fonctionnement des outils CI/CD et monitoring	14
4.1	Introduction	15
4.2	Présentation des outils	15
4.2.1	Git	15
4.2.2	Gitlab-Ci	15
4.2.3	Docker	16
4.2.4	Ansible	16
4.2.5	Kubernetes	17
4.2.6	Terraform	18
4.2.7	Prométhéus	18
4.2.8	Grafana	19
4.3	Etude comparative entre les outils	20
4.3.1	Outils de gestion des versions : Git vs SVN	20
4.3.2	Outils d'orchestration et déploiement des conteneurs : Kubernetes vs Docker swarm	22
4.3.3	Outils de gestion et configuration du cluster : Ansible vs Puppet vs Chef	24
4.3.4	Comparaison Kibana et Grafana	25
4.3.5	Outils d'intégration continue	26
4.3.6	Outils d'approvisionnement	26
4.4	Conclusion	27
5	Réalisation	28
5.1	Introduction	29
5.2	Environnement de Développement	29
5.2.1	Les variables de terraform	29
5.2.2	Provisionnement de la machine virtuelle	31
5.2.3	Déploiement Continue	40
5.3	L'environnement Préprod et Prods	43
5.3.1	Les variables Terraform	43
5.3.2	Provisionnement du Cluster	46
5.3.3	Mise en place des outils nécessaire	47

5.3.4	Déploiement Continue	56
5.3.5	Exécution de pipeline :	59
5.4	L'environnement production	60
5.4.1	Mise en place des outils	60
5.4.2	Déploiement sur le cluster	63
5.5	Monitoring	66
5.5.1	Installation des outils	66
5.5.2	Installation des outils	71
5.6	Conclusion	73

Table des figures

2.1	Logo Roundesk	4
2.2	Architecture de la solution actuelle	5
3.1	Architecture de la solution proposée	9
3.2	CI/CD Architecture	11
3.3	Virtualisation	12
3.4	Conteneurisation	13
4.1	Logo de Git	15
4.2	Logo gitlab	16
4.3	Logo Docker	16
4.4	Logo Ansible	17
4.5	Logo Kubernetes	18
4.6	Logo terraform	18
4.7	Logo Prometheus	19
4.8	Logo Grafana	19
5.1	fichier variables.tf	29
5.2	fichier terraform.tfvars	30
5.3	fichier outputs.tf	30
5.4	Main.tf (Partie 1)	31
5.5	Main.tf (Partie 2)	32
5.6	Main.tf (Partie 3)	33
5.7	Main.tf (Partie 4)	34
5.8	: Playbook.yml	35
5.9	: Role update-install-needs main.yml	36
5.10	: Role install-docker.yml	36
5.11	: Role install-nginx.yml	37

5.12	: Role install-mysql.yml	38
5.13	: Role config-db main.yml	39
5.14	: Le Résultat de terraform Apply	40
5.15	Dockerfile dev	41
5.16	Ansible Playbook	42
5.17	Gitlab-ci.yml	43
5.18	Fichier Variables.tf	44
5.19	Fichier Terraform.tvars	44
5.20	main.tf	46
5.21	Installation de MySQL Operator	47
5.22	Installation de MySQL Cluster	48
5.23	mysql-cluster.yml	49
5.24	mysql-secret.yml	49
5.25	Création et importation des données	50
5.26	registrysecret.yml	51
5.27	okidoki-svc.yml	52
5.28	Application des fichiers services et secret	52
5.29	Mise en place d'Nginx Ingress	53
5.30	Fichier Cluster-issuer.yml	54
5.31	ssl-ingress-pre-prod.yml - avant l'exécution du script	55
5.32	Test de déploiement	55
5.33	Fichier Gitlab-Ci.yml	56
5.34	: Dockerfile-prod	57
5.35	Oki-Doki-Deployment.yml	58
5.36	gitlab-ci.yml - stage de déploiement	59
5.37	Avancement de gitlab pipeline	60
5.38	MySQL-secret.yml	61
5.39	mysql-cluster-prod.yml	62
5.40	vérification de la mise en place de mysql-cluster pod	62
5.41	Mise en place de bases de données prod	63
5.42	l'environnement prod - vérification de la création du namespace prod	63
5.43	registry-secret.yml	63

5.44 okidoki-svc.yaml	64
5.45 Okidoki-deploy.yaml	64
5.46 ssl-ingress-prod.yaml	65
5.47 Interface de l'application déployée	66
5.48 Script de l'environnement pré-prod - installation du Prometheus	67
5.49 Pods du monitoring	67
5.50 Deployments du monitoring	68
5.51 Services du monitoring	68
5.52 Port-forward du prometheus	69
5.53 Port-forward du Grafana	69
5.54 Interface du Prometheus	69
5.55 Interface du Grafana	70
5.56 Configuration du Grafana	70
5.57 Liste des métriques à afficher	71
5.58 Utilisation de CPU et de Mémoire par le cluster	72
5.59 Charge CPU de l'environnement prod	72
5.60 Charge CPU de l'environnement pré-prod	73
5.61 Charge CPU - l'environnement prod vs l'environnement pré-prod	73

Liste des tableaux

4.1	Comparaison du « Git » et « SVN »	21
4.2	Comparaison Kubernetes et Docker Swarm	23
4.3	Tableau de comparaison des outils de configuration (Chef, Ansible et Puppet)	24
4.4	comparaison Grafana et Kibana	25
4.5	Comparaison de différents outils CI/CD	26
4.6	Comparaison Terraform et Azure	27

Chapitre 1

Introduction générale

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

Faire intervenir autant d'équipes peut mener à des conflits puisque l'objectif de chaque équipe est différent de l'autre. Alors que les développeurs souhaitent innover et faire évoluer les applications, l'équipe de production cherche, avant tout, à maintenir la stabilité du système informatique. D'ailleurs, chacun suit ses processus, et travaille avec ses propres outils qui sont rarement communicants. Par conséquent, les relations entre ces équipes peuvent être conflictuelles.

Ces différents génèrent des retards de livraison et des coûts supplémentaires pour l'entreprise qui n'ont pas été prévus au départ, avec un impact sur le client dont la satisfaction est le centre des préoccupations de l'entreprise. Il devient alors évident qu'il faut adopter une nouvelle approche qui permet d'unifier le processus de développement et celui de production afin d'éviter tous les problèmes cités précédemment.

De ce fait, la notion de DevOps est née. Il s'agit d'une approche qui se base sur la synergie entre la partie opérationnelle et la partie production.

L'alignement de l'ensemble des équipes du système d'information sur un objectif commun permet de réduire les conflits entre ces différents intervenants, d'éviter les retards dus

à la communication entre eux, et d'améliorer, ainsi, les délais de livraisons (Time-To-Market). C'est dans ce cadre que s'inscrit notre Projet de Fin d'Etude au sein de Roundesk Technologies il s'agit de mettre en place un processus d'Intégration continue/Livraison continue, Livraison continu pour l'application web « OKI-DOKI »

Chapitre 2

Cadre général du projet

2.1 Introduction

Le premier chapitre a pour objectif de situer le projet dans son cadre général en présentant dans une première partie l'organisme d'accueil au sein duquel nous avons effectué notre stage de PFE. Dans la deuxième partie nous introduisons le sujet tout en faisant une étude de l'existant et une évaluation indépendante et neutre. Ensuite nous présentons la problématique qui a engendré ce travail ainsi que la solution proposée

2.2 Présentation de Roundesk Technologies

Roundesk Technologies : Roundesk Technologies, Startup qui s'adresse aux auto-entrepreneurs, PME (Petite ou moyenne entreprise) / PMI (petite ou moyenne industrie) / ETI (Entreprise de taille intermédiaire) et aux grandes entreprises, offre 'a ses clients la meilleure qualité en développement des outils CRM- ERP -GRC.



FIG. 2.1 – Logo Roundesk

2.3 Objectifs

Notre objectif à travers ce projet étant d'abord d'adapter à la culture DevOps en appliquant les bonnes pratiques de cette nouvelle approche de développement logiciel sur le projet OkiDoki. Roundesk Technologies vise à changer sa stratégie en passant vers cette nouvelle stratégie qui a comme objectifs d'accélérer et automatiser les différentes étapes du cycle de vie des logiciels pour garantir des logiciels de qualité et donc une meilleure satisfaction des utilisateurs finaux. De plus, nous avons comme objectif d'atteindre une meilleure version du projet OkiDoki à travers la mise en œuvre d'un système de déploie-

ment continue et de monitoring, un système qui collecte les journaux des évènements et les métriques d'infrastructure et affiche ce qui est essentiel à la prise de décision.

2.4 Analyse de l'existant

Actuellement, l'équipe de développement travaille avec la méthode AGILE SCRUM mais sans avoir de rapport de qualité sur leurs codes. L'équipe opérationnelle intervient après afin de livrer une version finale sans avoir contrôlé le code. On ne trouve aucun moyen de contrôle ou de monitoring pour ce processus, alors l'automatisation de ces tâches devient une nécessité ainsi que la synchronisation des équipes vu que le temps gaspillé est remarquable et il a un impact sur la productivité.

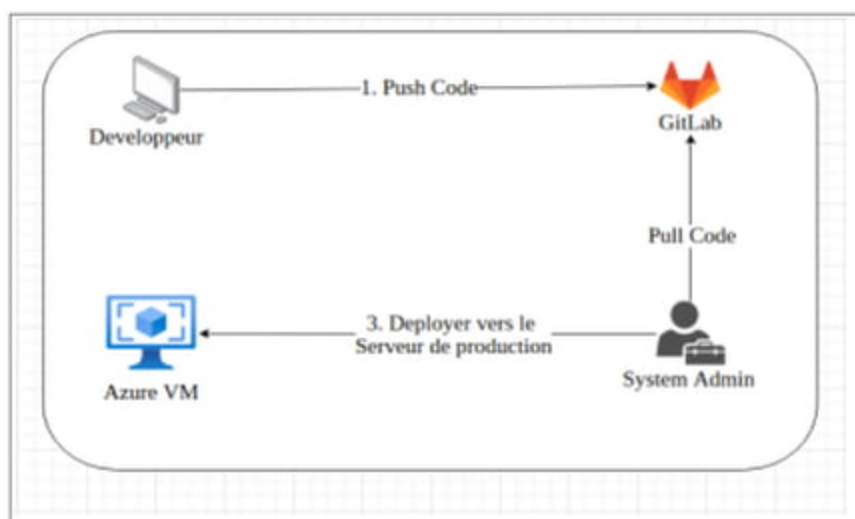


FIG. 2.2 – Architecture de la solution actuelle

2.5 Problématique

Le temps du cycle d'un projet, la date de livraison ainsi que la qualité de code sont des facteurs importants pour chaque entreprise. On trouve dans certaines entreprises des difficultés pour la gestion des équipes spécialement la relation entre l'équipe de développement et l'équipe opérationnelle (administrateur système)

L'objectif principal d'une équipe d'administrateurs est de garantir la stabilité des systèmes. La meilleure manière d'atteindre cet objectif est de contrôler sévèrement la qualité des changements qui sont apportés aux systèmes qu'ils maintiennent. De son côté, l'équipe développement a pour objectif d'apporter les changements nécessaires au moindre coût et le plus vite possible. Par suite, cette séparation des charges entre les deux types d'équipes a rapidement mené à un conflit perpétuel du fait de l'incompatibilité des objectifs respectifs. Ceci peut être illustré en considérant les 3 contraintes de la gestion de projet qui sont le coût, la qualité et le temps

2.6 Solution proposée

Le souci primordial serait donc d'avoir un procédé qui permet d'automatiser le processus d'intégration, des tests et de contrôle de qualité de code ainsi que la livraison continue pour les applications. D'où la nécessité d'une méthode permettant, dans un premier temps, de concevoir et de déployer rapidement une application de qualité, puis de faire en sorte que des modifications plus ou moins importantes puissent être disponibles en quelques heures ou quelques jours. Donc les défis de mon projet sont l'automatisation des tests et d'intégration de codes ainsi que la livraison continue pour les clients. L'objectif de mon projet consiste en la mise en place d'un pipeline d'intégration et de déploiement continu, ainsi que la notification des métriques de performances du pipeline précédemment mis en place. Ce projet englobe 3 grands axes :

- Intégration continue.
- Déploiement continu.
- Notification et reporting.

2.6.1 Conclusion

Au cours de ce premier chapitre, j'ai commencé par présenter l'organisme d'accueil Roundesk Technologies. Par la suite j'ai étudié l'existant en citant la problématique. Enfin, j'ai expliqué la solution proposée en présentant mes objectifs Dans le chapitre suivant, je détaillerai, clairement, la notion de DevOps, la virtualisation traditionnelle et la conteneurisation, ensuite je ferai une approche sur l'intégration continue/ le déploiement continue et enfin je citerai les outils utilisés

Chapitre 3

Intégration continue/Livraison
continue, Déploiement continu

3.1 Introduction

Ce chapitre a pour but de présenter, dans son ensemble, l'intégration continue, la livraison continue, le déploiement continu, le concept de virtualisation et conteneurisation et de définir le DevOps. La figure ci-dessous représente l'architecture globale de mon projet.

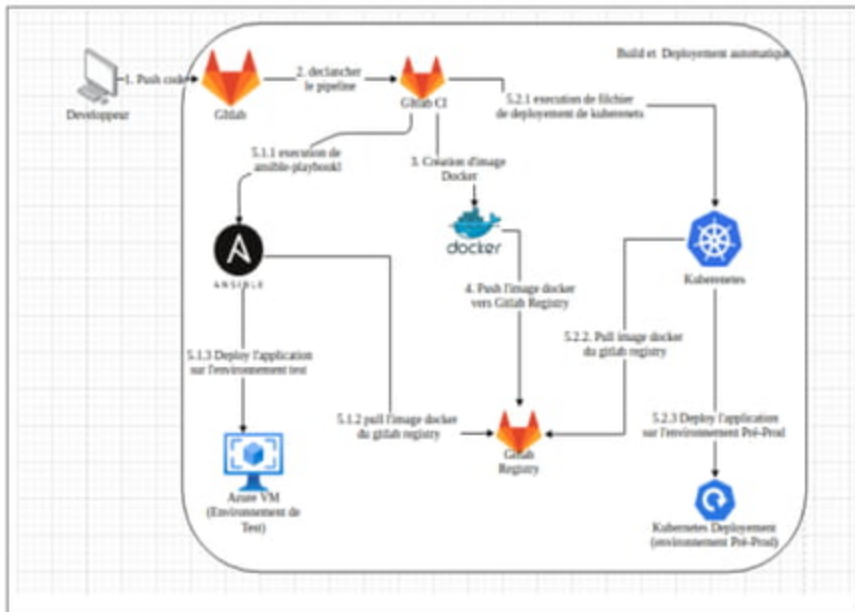


FIG. 3.1 – Architecture de la solution proposée

3.2 Approche intégration, livraison déploiement continus

L'approche CI/CD représente une solution aux problèmes posés par l'intégration de nouveaux segments de code pour les équipes de développement et d'exploitation. Elle permet d'augmenter la fréquence de distribution des applications grâce à l'introduction de l'automatisation au niveau des étapes de développement des applications. Les principaux concepts liés à l'approche CI/CD sont l'intégration continue, la livraison continue et le déploiement continu.[1]

3.2.1 Intégration continue (CI)

Les développeurs pratiquant l'intégration continue fusionnent leurs modifications dans la branche principale aussi souvent que possible. Les modifications du développeur sont validées en créant une version et en exécutant des tests automatisés sur la version. Ce faisant, on évite l'enfer de l'intégration qui survient généralement lorsque les utilisateurs attendent le jour de publication pour fusionner leurs modifications dans la branche de publication. L'intégration continue insiste beaucoup sur l'automatisation des tests pour vérifier que l'application n'est pas endommagée à chaque fois que de nouveaux commits sont intégrés dans la branche principale.[1]

3.2.2 La livraison continue/déploiement continu (CD)

La livraison continue est une extension de l'intégration continue qui permet de publier rapidement de nouvelles modifications pour les clients et de manière fiable. Cela signifie qu'en plus d'avoir automatisé les tests, on pourra aussi automatiser le processus de publication en déployant une application à tout moment en cliquant sur un bouton. En théorie, avec une livraison continue, on peut décider de publier à tout moment ou selon les besoins. Toutefois, si on souhaite réellement tirer parti des avantages de la livraison continue, on doit effectuer la mise en production dès que possible afin d'être sûr de libérer de petits lots faciles à résoudre en cas de problème.

Le déploiement continu va plus loin que la livraison continue. Avec cette pratique, chaque modification qui passe par toutes les étapes du pipeline de production est transmise aux clients. Il n'y a aucune intervention humaine, et seul un test ayant échoué empêchera le déploiement d'un nouveau changement en production.

Le déploiement continu est un excellent moyen d'accélérer la boucle de feedback avec les clients et de soulager l'équipe, car il n'y a plus de publication. Les développeurs peuvent se concentrer sur la création de logiciels et voient leur travail mis en ligne quelques minutes après l'avoir terminé [2]

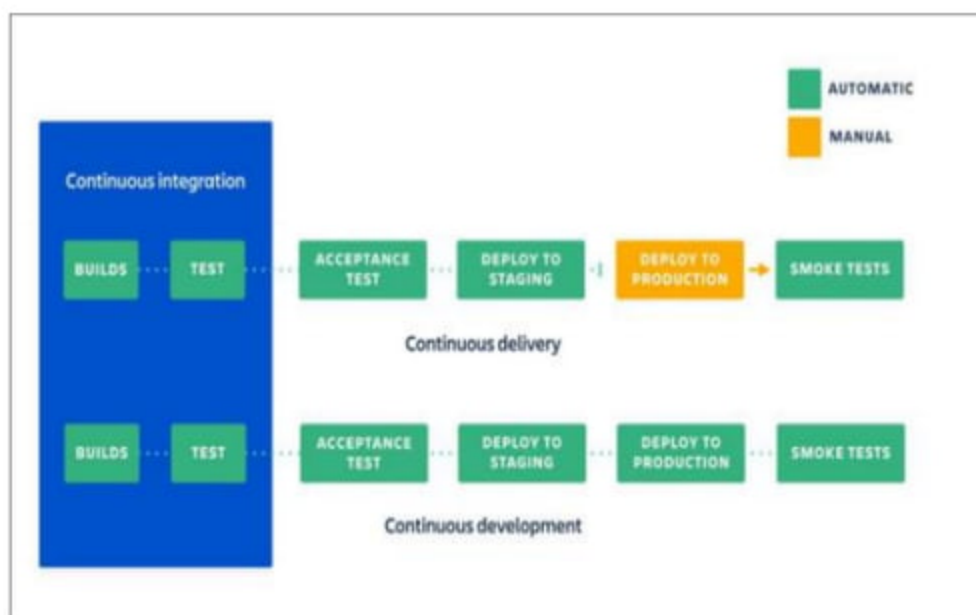


FIG. 3.2 – CI/CD Architecture

3.3 Définition de DevOps

DevOps est un ensemble de pratiques qui automatisent les processus entre les deux équipes : Celle du développement et celle des administrateurs système afin de leur permettre de développer, tester et livrer des logiciels plus rapidement et avec plus de fiabilité. Le concept de DevOps repose sur la mise en place d'une culture de collaboration entre des équipes qui étaient, historiquement, cloisonnées. L'accent est mis sur le changement de mentalité, la collaboration accrue et une intégration plus poussée.

DevOps associe la méthodologie Agile, l'automatisation, la livraison continue et bien plus encore pour aider les équipes de développement et l'équipe administration système à gagner en efficacité, à innover plus rapidement et à offrir plus de valeur ajoutée aux business et aux clients.

3.4 Approche virtualisation / conteneurisation

La virtualisation et la conteneurisation sont deux technologies très utilisées dans le déploiement alors on va présenter ces concepts.

3.4.1 La virtualisation

[3] La virtualisation est une technologie utilisée pour permettre le fonctionnement de plusieurs machines virtuelles disposant chacune de son système d'exploitation spécifique partageant la même infrastructure physique. Une machine virtuelle (VM) contient un système d'exploitation complet, avec ses pilotes, fichiers binaires ou bibliothèques. Chaque VM s'exécute sur un hyperviseur (par exemple : VMWare, VirtualBox, ...) : C'est un logiciel de virtualisation qui est directement installé sur le système d'exploitation principal ou hôte. Il permet de créer plusieurs environnements clos et indépendants qui pourront à leur tour héberger d'autres systèmes d'exploitation aussi appelés systèmes invités. Les environnements indépendants ainsi créés grâce à l'hyperviseur sont des machines virtuelles.

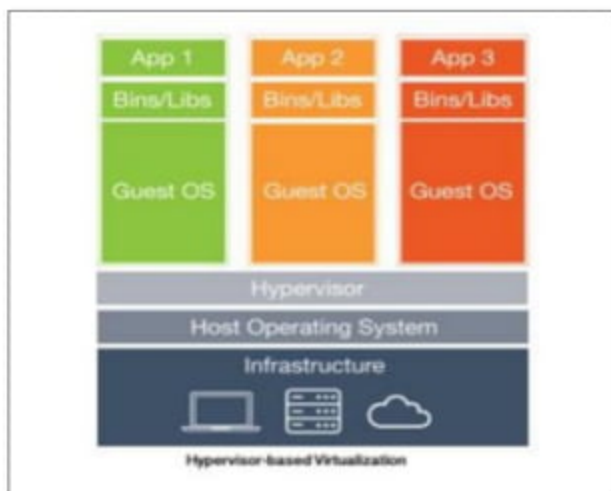


FIG. 3.3 – Virtualisation

3.4.2 La Conteneurisation

Le concept de conteneurisation permet d'exploiter le noyau (kernel) du système hôte : Il n'est donc pas nécessaire, comme pour les machines virtuelles, d'installer un nouveau système d'exploitation. Cette approche réduit le gaspillage des ressources car chaque conteneur ne renferme que l'application et les fichiers binaires ou bibliothèques associées. On utilise donc le même système d'exploitation (OS) hôte pour plusieurs conteneurs. La

conteneurisation rend donc le déplacement d'application virtuelle plus simple entre des systèmes d'exploitation identiques et demande moins de ressources de stockage, de RAM, de CPU, etc. Comme outils de conteneurisation, on peut citer Lxc et surtout Docker la solution de conteneurisation la plus utilisée aujourd'hui.

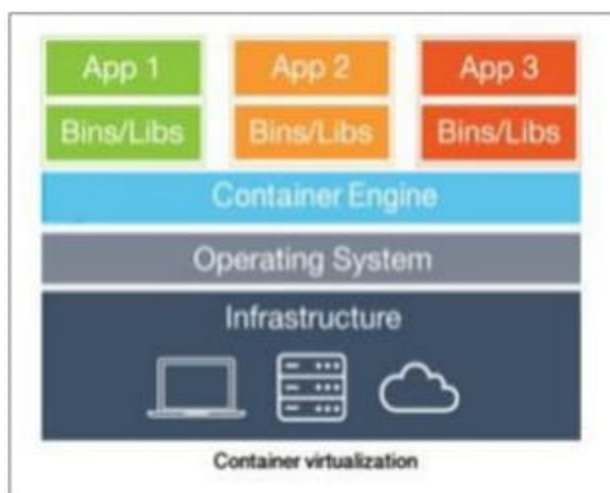


FIG. 3.4 – Conteneurisation

3.5 Conclusion

Au cours de ce chapitre, on a présenté les concepts de base de ce projet, et définis le concept de virtualisation et conteneurisation, ensuite on a défini la notion DevOps et enfin on a mis l'accent sur l'intégration continue, la livraison et le déploiement continue.

Chapitre 4

Architecture et principe de
fonctionnement des outils CI/CD et
monitoring

4.1 Introduction

Pour la mise en place de différents environnements et la configuration du CI/CD on va introduire chaque outil exploité et utilisée dans notre solution.

4.2 Présentation des outils

4.2.1 Git

Git [4] est un système de contrôle de version open source. Concrètement, c'est un outil qui te permet de traquer tous les fichiers de ton projet. Chaque modification de fichier est alors détectée par Git et versionnée dans une version instantanée. Un historique de modification va être disponible sur le projet. On peut consulter et même si retourner en arriere dans les changements.

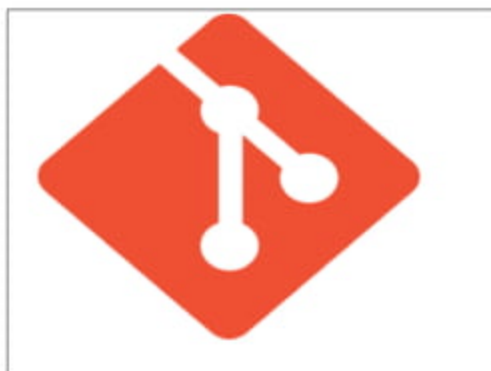


FIG. 4.1 – Logo de Git

4.2.2 Gitlab-Ci

Le service GitLab CI (intégration continue) [5] fait partie de GitLab qui construit et teste le logiciel chaque fois que le développeur fait un « push » sur le code de l'application. GitLab CD (Déploiement continu) est un service logiciel qui place les modifications de chaque code dans la production, ce qui entraîne un déploiement quotidien de la production.



FIG. 4.2 – Logo gitlab

4.2.3 Docker

Docker [6] permet d'embarquer une application dans un ou plusieurs containers logiciels qui pourra s'exécuter sur n'importe quel serveur machine, qu'il soit physique ou virtuel. Docker fonctionne sous Linux comme Windows Server. C'est une technologie qui a pour but de faciliter les déploiements d'application, et la gestion du dimensionnement de l'infrastructure sous-jacente. Elle est proposée par la société Docker, en partie en open source.



FIG. 4.3 – Logo Docker

4.2.4 Ansible

Un parmi les outils d'automatisation, le plus performant, largement utilisé par les administrateurs pour l'automatisation de la configuration et de la gestion des serveurs.

Les processus Ansible [7] utilisent un format YAML pour définir les différentes tâches à implémenter. Le regroupement de ces tâches en relation avec d'autres tâches qui seront exécutées ainsi la définition des variables et des listes d'hôtes servira à la définition d'un autre fichier YAML appelé Playbook.



FIG. 4.4 – Logo Ansible

4.2.5 Kubernetes

Kubernetes [8] (connu aussi sous le nom de k8s ou « kube ») est une plateforme d'orchestration de conteneurs open source qui automatise plusieurs processus manuels impliqués dans le déploiement, l'évolution, l'entretien, la planification et l'exploitation de nombreux conteneurs d'application sur un cluster. Il contient des outils d'orchestration et d'équilibrage de charge pour une utilisation avec les conteneurs Docker. Pods est l'unité de base de l'ordonnancement dans Kubernetes.



FIG. 4.5 – Logo Kubernetes

4.2.6 Terraform

Terraform [9] est un outil d'infrastructure en tant que code. C'est open source et écrit en « Go » par « Hashicorp ». Concrètement, ça permet de déclarer, via code, ce qu'on veut pour l'infrastructure souhaité. Dans des fichiers de configuration structurés on peut pouvoir manager automatiquement l'infrastructure souhaité sans intervention manuelle.



FIG. 4.6 – Logo terraform

4.2.7 Prométhéus

Prometheus [10] est un outil de collecte de métriques et d'alerte développé et publié en open source par « SoundCloud ». Prometheus est de conception similaire au système de surveillance « Borgmon » de « Google », et un système relativement modeste peut gérer

la collecte de centaines de milliers de métriques chaque seconde. Correctement réglé et déployé, un cluster Prometheus peut collecter des millions de métriques chaque seconde.



FIG. 4.7 – Logo Prometheus

4.2.8 Grafana

Grafana [11] est un outil d'affichage de données de toutes sortes, et de sources différentes. Il peut être utilisé comme un tableau de bord pour chaque section d'une entreprise, afin d'obtenir des données sur les utilisateurs tout en respectant le RGPD (bien entendu). Grafana, propose aussi des alertes au utilisateurs (par slack, mails etc.) .



FIG. 4.8 – Logo Grafana

4.3 Etude comparative entre les outils

Dans cette partie, on explore la différence entre les outils courants utilisés sur le marché et pourquoi nous avons fait le choix pour chaque outil selon notre besoin fonctionnel.

4.3.1 Outils de gestion des versions : Git vs SVN

[12] Les deux outils de gestion de version les plus populaires sur le marché sont Git et SVN. Ils permettent le flux de travail et la gestion de projets dans le développement de logiciels, voici un tableau (voir tableau 4.1) qui compare ces deux outils qui permette de faire le bon choix.

Critères	Git	SVN
Architecture	Distribué, tous les développeurs qui vérifient le code d'un repository/serveur central auront leur propre clone. Il agit à la fois comme serveur et client.	Nécessite un serveur centralisé et un client utilisateur. Seul le repository central a l'historique complet des changements. Les utilisateurs doivent communiquer via le réseau avec le répertoire central pour Obtenir l'historique
Contrôle d'accès et autorisations	Aucun "Commit Access" requis puisque le système est distribué. Il suffit juste de décider du contenu à fusionner à partir de quel Utilisateur. .	Besoin d'accès commit en raison de la centralisation.
Stockage	Stocker tout le contenu dans un répertoire « . git », c'est le dépôt du code cloné sur la machine cliente.	Stockage de métadonnées de fichiers en tant que dossier caché « . svn »
Révision	Un outil SCM (gestion de code source), pas de fonction de numéro de révision globale.	Un système de contrôle des révisions, comporte un numéro de révision global

TAB. 4.1 – Comparaison du « Git » et « SVN »

D'après ce tableau comparatif, on a décidé d'utiliser « Git »

4.3.2 Outils d'orchestration et déploiement des conteneurs : Kubernetes vs Docker swarm

[13]Kubernetes et Docker swarm sont deux outils qui peuvent être utilisés pour gérer le cycle de vie de conteneurs. Le tableau ci-dessous illustre une comparaison entre Kubernetes et Docker swarm.

Critères	Kubernetes	Docker Swarm
Evolutivité	La mise à l'échelle et le déploiement des conteneurs sont ralentis car il est une infrastructure tout-en-un	Déployer les conteneurs très rapidement.
Haute disponibilité	Les pods de kubernetes sont répartis entre les noeuds ce qui offre une haute disponibilité en tolérant l'échec de l'application. Détection des conteneurs crashés, il essaie encore de le redéployer dans le même noeud.	Les conteneurs peuvent être reproduits dans les noeuds swarm, il offre alors une haute disponibilité.
Équilibrage de charge	L'activation de l'équilibrage de charge nécessite une configuration de service manuelle.	Équilibrage de charge interne automatisé via n'importe quel noeud du cluster.
Administration du cluster	CLI est l'API REST qui garantit une flexibilité pour gérer le cluster. Le contrôle et la surveillance des états de noeud sont assurés via un tableau de bord.	CLI pour interagir avec le cluster mais pas de tableau de bord.
Installation et la configuration du cluster	Complexe	Simple et rapide

TAB. 4.2 – Comparaison Kubernetes et Docker Swarm

Suite à cette comparaison entre Kubernetes et Docker swarm, on a choisi d'utiliser Kubernetes pour orchestrer et déployer les conteneurs puisqu'il est le plus répandue dans le marché ainsi que l'automatisation de la montée en charge des conteneurs et la supervision des nœuds et des conteneurs via un tableau de bord

4.3.3 Outils de gestion et configuration du cluster : Ansible vs Puppet vs Chef

La mise en place de serveurs peut être une tâche fastidieuse qui est répétitive et sujette à des erreurs de traitement. L'apparition de "Chef", "Puppet" et "Ansible" était attendue depuis longtemps. Grâce à ces outils, il n'est plus nécessaire de configurer chaque station individuellement, il suffit de démarrer un script, et des dizaines, des centaines ou des milliers de machines sont configurées en même temps, réduisant ainsi l'effort fourni et les risques d'erreur dus à la manipulation. Le tableau ci-dessous compare chacun de ses 3 outils [14]

Critères	Ansible	Chef	Puppet
Système de Communication	Rapide	Très lent	Lent
Exécution de Configuration	Facile	Difficile	Difficile
Langage de configuration	YAML : permet de représenter des Données structurées	DSL(Ruby)	DSL(Puppets) : Propre à Puppet
Installation	Facile	Complicé	Complicé
Architecture	Client	Client/serveur	Client/serveur

TAB. 4.3 – Tableau de comparaison des outils de configuration (Chef, Ansible et Puppet)

Suite à la comparaison entre Chef, Puppet et Ansible, on a choisi d'utiliser Ansible comme outil de gestion et configuration du cluster puisqu'il n'est pas nécessaire de l'installer côté client.

4.3.4 Comparaison Kibana et Grafana

[15]Grafana et Kibana sont deux outils de visualisation de données bien connus des équipes IT. Ils permettent de détecter des tendances dans le comportement des infrastructures et jouent un rôle crucial dans la stratégie de surveillance appliquée par une DSI.

GRAFANA	KIBANA
Outil d'analyse et de surveillance de journaux autonomes open source	Fait partie de la pile ELK, utilisée pour l'analyse des données et la surveillance des journaux
Multiplateforme, offre une intégration avec diverses plates-formes et bases de données.	Pas multiplateforme, il est spécialement conçu pour la pile ELK. Le K en ELK est pour Kibana
Prend en charge InfluxDB, AWS, MySQL, PostgreSQL et bien d'autres.	Prend en charge Elasticsearch.
Mieux adapté aux applications qui nécessitent des mesures de surveillance en temps réel comme la charge du processeur, la mémoire, etc.	Pas multiplateforme, il est spécialement conçu pour la pile ELK. Le K en ELK est pour Kibana
Alertes personnalisées en temps réel à mesure que les données arrivent	Alertes uniquement à l'aide de plugins, prend en charge des API
Fournit une plate-forme pour utiliser plusieurs éditeurs de requêtes	Prend en charge la syntaxe Lucene, le DSL et les requêtes Elasticsearch
Les variables d'environnement sont configurées via le fichier .ini.	La configuration se fait dans des fichiers YAML
Configuration très simple car elle est autonome	Doit être configuré par rapport à la même version du nœud élastique
Utilisé par : 9gag, Digitalocean, postmates, etc.	Utilisé par : trivago, bitbucket, Hubspot, etc.

TAB. 4.4 – comparaison Grafana et Kibana

D'après cette comparaison on a déduit que le meilleur choix pour notre cas sera Grafana

4.3.5 Outils d'intégration continue

[16] Après une recherche approfondie, et pour obtenir un bref aperçu des outils Ci, on a jugé utile Gitlab-Ci, Jenkins, CircleCi, TeamCi et Bamboo dans un tableau présenté ci-dessous (voir tableau 4.5)

	Jenkins	CircleCi	TeamCity	Bamboo	Gitlab Ci
Open source	Oui	Non	Non	Non	Non
Installation	Intermédiaire	Intermédiaire	Intermédiaire	Intermédiaire	Intermédiaire
Fonctionnalité intégré	3/5	4/5	4/5	4/5	4/5
Tarifs	Gratuit	39 USD par mois	299 USD	10 USD	4 USD par utilisateur
Version gratuite	Oui	Oui	Oui	Oui	Oui
S.E supporté	Windows, Linux, MacOS	Linux, MacOS	Windows, Linux, MacOS, Solaris	Windows, MacOS, Solaris	Linux, Debian, Ubuntu, CentOS, Oracle linux

TAB. 4.5 – Comparaison de différents outils CI/CD

Suite à cette comparaison entre les différents outils sur le marché, on a décidé d'utiliser Gitlab-ci vu qu'il possède plusieurs fonctionnalités intégrées, supporte sur plusieurs plateformes et à un prix raisonnable (version Enterprise)

4.3.6 Outils d'approvisionnement

[17] Les deux outils sont conçus pour résoudre des problèmes similaires, leur objectif général est de simplifier et de faciliter la mise en place de l'infrastructure, le tableau ci-dessous contient une étude comparative entre ces deux outils.

Critères	Azure Resource manager (ARM)	Terraform
Langage	JSON	HashiCorp Config Language (HCL)
Multi Cloud	NON	OUI
rapidité	Prend du temps	Plus rapide que ARM
Validation Avant le déploiement	OUI (az group deployment validate)	OUI (Terraform plan)
Compréhensibilité des messages d'erreur	Plus lisible que Terraform	un peu difficile
Maintenabilité	OUI	plus facile à maintenir

TAB. 4.6 – Comparaison Terraform et Azure

Donc, les deux outils semblent plutôt bons en matière de codage d'infrastructure pour Azure, nous avons choisi Terraform puisqu'il est multi cloud tandis que ARM est seulement utilisé par Azure.

4.4 Conclusion

Au cours de ce chapitre, on a présenté les choix concernant les outils utilisés qui seront le noyau de la partie réalisation, ainsi que la comparaison et justification de notre choix.

Chapitre 5

Réalisation

5.1 Introduction

Au cours de ce dernier chapitre, on va exposer le travail accompli, explorer les différentes étapes pour réaliser le travail final en détaillant chaque étape et en incluant quelques captures d'écran expliquant le processus de déploiement de la solution

5.2 Environnement de Développement

Pour l'environnement de développement, et à cause de coups on a choisi de créer une machine virtuelle avec la configuration nécessaire pour exécuter le projet.

5.2.1 Les variables de terraform

```
variable "resource_group_name" {  
  type      = string  
  description = "RG name in Azure"  
}  
  
variable "resource_group_location" {  
  type      = string  
  description = "RG location in Azure"  
}  
  
variable "virtual_network_name" {  
  type      = string  
  description = "VNET name in Azure"  
}  
  
variable "subnet_name" {  
  type      = string  
  description = "Subnet name in Azure"  
}  
  
variable "public_ip_name" {  
  type      = string  
  description = "Public IP name in Azure"  
}  
  
variable "network_security_group_name" {  
  type      = string  
  description = "NSG name in Azure"  
}  
  
variable "network_interface_name" {  
  type      = string  
  description = "NIC name in Azure"  
}  
  
variable "linux_virtual_machine_name" {  
  type      = string  
  description = "Linux VM name in Azure"  
}  
  
variable "linux_virtual_machine_size" {  
  type      = string  
  description = "Linux VM size in Azure"  
}
```

FIG. 5.1 – fichier variables.tf


```
resource_group_name      = "okidoki-vm"
resource_group_location  = "francecentral"
virtual_network_name     = "okidokivn"
subnet_name              = "subnet"
public_ip_name           = "publicip"
network_security_group_name = "nsg"
network_interface_name   = "nic"
linux_virtual_machine_name = "Okidokivm"
linux_virtual_machine_size = "Standard_DS1_v2"
```

FIG. 5.2 – fichier terraform.tfvars

```
* outputs.tf X
vm2 > outputs.tf > ...
1  output "vm_ip" {
2    |   value = azurerm_linux_virtual_machine.linuxvm.public_ip_address
3    | }
4
5  |
```

FIG. 5.3 – fichier outputs.tf

- Variables.tf (Voir figure5.1) est utilisé pour définir le type de variables et éventuellement définir une valeur par défaut
- Terraform.tfvars (Voir figure5.2) contient toutes les valeurs des variables présentées dans le fichier variables.tf
- Outputs.tf (Voir figure5.20) contient les résultats attendus à afficher lors de la terminaison de la création (dans notre cas, l'adresse IP publique de la machine virtuelle)

5.2.2 Provisionnement de la machine virtuelle

Dans cette partie on présente le fichier `main.tf` de terraform qui est responsable du provisionnement et la création de la machine virtuelle, on présente chaque block en expliquant son rôle.

```
provider "azurerm" {  
  version = "=2.37.0"  
  features {}  
}  
  
# Create a resource group if it doesn't exist  
resource "azurerm_resource_group" "rg" {  
  name       = var.resource_group_name  
  location   = var.resource_group_location  
}  
  
# Create virtual network  
resource "azurerm_virtual_network" "vnet" {  
  name                = var.virtual_network_name  
  address_space       = ["10.0.0.0/16"]  
  location             = azurerm_resource_group.rg.location  
  resource_group_name = azurerm_resource_group.rg.name  
}  
  
# Create subnet  
resource "azurerm_subnet" "subnet" {  
  name                 = var.subnet_name  
  resource_group_name = azurerm_resource_group.rg.name  
  virtual_network_name = azurerm_virtual_network.vnet.name  
  address_prefixes     = ["10.0.1.0/24"]  
}  
  
# Create public IPs  
resource "azurerm_public_ip" "public_ip" {  
  name                 = var.public_ip_name  
  location             = azurerm_resource_group.rg.location  
  resource_group_name = azurerm_resource_group.rg.name  
  allocation_method    = "Dynamic"  
}
```

FIG. 5.4 – Main.tf (Partie 1)

- **Provider** : le fournisseur Cloud choisi (Azure pour notre cas) .
- **azurerm-resource-group** : Crée un groupe de ressource avec une région choisie dans la liste des régions azure.

- **azurerm-virtual-network** : Crée un réseau virtuel dans la région sélectionné pour le groupe de ressource sélectionné avec une marge souhaité
- **azurerm-subnet** : Crée un sous réseau dans le réseau virtuel déjà crée dans l'étape précédente, dans le groupe de ressource déjà crée
- **azurerm-public-ip** : Crée une adresse IP publique et l'ajouter au groupe de ressource

```
resource "azurerm_network_security_group" "nsg" {
  name                = var.network_security_group_name
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  security_rule {
    name                = "SSH"
    priority            = 1001
    direction          = "Inbound"
    access              = "Allow"
    protocol            = "Tcp"
    source_port_range   = "*"
    destination_port_range = "22"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }

  security_rule {
    name                = "p80"
    priority            = 1000
    direction          = "Inbound"
    access              = "Allow"
    protocol            = "Tcp"
    source_port_range   = "*"
    destination_port_range = "80"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }
}

# Create network interface
resource "azurerm_network_interface" "nic" {
  name                = var.network_interface_name
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name

  ip_configuration {
    name                = "myNicConfiguration"
    subnet_id          = azurerm_subnet.subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id = azurerm_public_ip.public_ip.id
  }
}
```

FIG. 5.5 – Main.tf (Partie 2)

- **azurerm-network-interface** : Créer une interface réseau permettant de créer un lien entre la machine virtuelle et le réseau virtuelle déjà crée.

- **azurerm-network-security-group** : Créer un groupe de sécurité dans la ressource groupe et la région déjà choisis, cette partie permet de sécuriser tous les ports et ouvrir les ports 22 et 80 (pour nginx et ssh).

```
# Connect the security group to the network interface
resource "azurerm_network_interface_security_group_association" "association" {
  network_interface_id      = azurerm_network_interface.nic.id
  network_security_group_id = azurerm_network_security_group.nsg.id
}

# Generate random text for a unique storage account name
resource "random_id" "randomId" {
  keepers = {
    # Generate a new ID only when a new resource group is defined
    resource_group = azurerm_resource_group.rg.name
  }

  byte_length = 8
}

# Create storage account for boot diagnostics
resource "azurerm_storage_account" "storage" {
  name                = "diag${random_id.randomId.hex}"
  resource_group_name = azurerm_resource_group.rg.name
  location             = azurerm_resource_group.rg.location
  account_tier         = "Standard"
  account_replication_type = "LRS"
}
```

FIG. 5.6 – Main.tf (Partie 3)

- **azurerm-network-interface-security-group-association** : Permet d'associer le groupe de sécurité avec l'interface réseau créée.
- **Random-id** : Créer un texte aléatoire qui sera utilisé comme identifiant pour un autre composant (pour notre cas le stockage)

- **azurerm-storage-account** : Créer un espace de compte azure pour le stockage de différents types de fichiers (pour notre cas sauvegarde de diagnostics boot).

```
# Create virtual machine
terraform
resource "azurerm_linux_virtual_machine" "linuxvm" {
  name                = var.linux_virtual_machine_name
  location             = azurerm_resource_group.location
  resource_group_name = azurerm_resource_group.name
  network_interface_ids = [azurerm_network_interface.nic.id]
  size                = var.linux_virtual_machine_size

  os_disk {
    name           = "myOsDisk"
    caching        = "ReadWrite"
    storage_account_type = "Premium_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "UbuntuServer"
    sku      = "18.04-LTS"
    version  = "latest"
  }

  computer_name      = "myvm"
  admin_username     = "azureuser"
  disable_password_authentication = true

  admin_ssh_key {
    username   = "azureuser"
    public_key = "${file("azure_vm.pub")}"
  }

  provisioner "local-exec" {
    command = "sed -i 's/10.0.0.1/${azurerm_linux_virtual_machine.linuxvm.public_ip_address}/' ansible/inventory"
  }

  provisioner "local-exec" {
    command = "sleep 30 && ansible-playbook -i ansible/inventory ansible/playbook.yml --user azureuser --private-key=/skidoki_em/vml/azure_vm.pem"
  }
}
```

FIG. 5.7 – Main.tf (Partie 4)

- **azurerm-linux-virtual-machine** : Le bloc principal permettant de créer la machine virtuelle
- **os-disk** la configuration principal de disque
- **Source-image-reference** : la version de Linux souhaité installé
- **admin-ssh-key** : Permet de créer un utilisateur avec la clé ssh souhaité
- **boot-diagnostics** : associé le stockage déjà créé avec la machine pour sauvegarder les logs de boot

- **local-exec** : permet d'exécuter des commandes linux sur la machine lors de la création (dans notre cas sa nous permettra d'executer ansible playbook).

```
---
- hosts: azurevm
  become: true
  roles:
    - update-install-needs
    - install-docker
    - install-nginx
    - install-mysql
    - configure-db
```

FIG. 5.8 – : Playbook.yml

La figure ci-dessus représente le fichier `playbook.yml`, qui sera exécuté avec « Ansible » permettant d'installer et mettre en place la configuration nécessaire dans notre machine virtuelle, notre composant est partagé sur plusieurs rôles chaque rôle prend en charge de l'installation d'un module nécessaire.

```

name: Update apt repo
apt: update_cache=yes force_apt_get=yes cache_valid_time=3600

name: Install needs
apt:
  name:
    - apt-transport-https
    - ca-certificates
    - curl
    - gnupg
    - lsb-release
    - python-pip
  state: present
  autoclean: yes

```

FIG. 5.9 – : Role update-install-needs main.yml

a figure ci-dessus représente le fichier principal de la rôle « update-install-needs » ce rôle permet d'installer les bibliothèques essentielles dans notre machine en utilisant le module « apt ».

```

- name: Install docker python package
  pip:
    name: docker-py

- name: Add Docker GPG apt Key
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present

- name: Add Docker Repository
  apt_repository:
    repo: deb https://download.docker.com/linux/ubuntu bionic stable
    state: present

- name: Update apt and install docker-ce
  apt: update_cache=yes name=docker-ce state=latest

```

FIG. 5.10 – : Role install-docker.yml

La figure ci-dessus représente le fichier principal de la rôle « install-docker » ce rôle

permet d'installer l'environnement docker dans notre machine en utilisant les module « pip » et « apt ».

```
- name: Install nginx
  apt:
    name: nginx
    state: present

- name: Copy nginx conf file
  copy:
    src: ../default.conf
    dest: /etc/nginx/sites-enabled/default
    mode: '0644'

- name: Restart nginx
  service:
    name: nginx
    state: restarted
```

FIG. 5.11 – ; Role install-nginx.yml

La figure ci-dessus représente le fichier principal de la rôle « install-nginx » ce rôle permet d'installer le serveur web nginx et copier le fichier de configuration dans le répertoire nécessaire avec les modules « apt » et « copy ».

```
- name: Add specified repository into sources list
  apt_repository:
    repo: deb http://kr.archive.ubuntu.com/ubuntu xenial main
    state: present

- name: Add specified repository into sources list
  apt_repository:
    repo: deb http://archive.ubuntu.com/ubuntu trusty universe
    state: present

- name: Update apt-get repo and cache
  apt: update_cache=yes force_apt_get=yes

- name: Set MySQL root password
  debconf:
    name: mysql-server-5.6
    question: mysql-server/root_password
    value: "root"
    vtype: password

- name: Confirm MySQL root password
  debconf:
    name: mysql-server-5.6
    question: mysql-server/root_password_again
    value: "root"
    vtype: password

- name: install mysql
  apt:
    name: mysql-server-5.6

- name: create mysql.sock file
  file:
    path: /var/run/mysqld/mysql.sock
    owner: mysql
    group: mysql
    state: touch

- name: Restart mysql
  service:
    name: mysql
    state: restarted
```

FIG. 5.12 – : Role install-mysql.yml

La figure ci-dessus représente le fichier principal de la rôle « install-mysql » ce rôle permet d'installer le serveur base de données MYSQL et copier le fichier de configuration dans le répertoire nécessaire avec les modules « apt » et « copy ».

```
- name: t1
  shell:
    mysql -uroot -proot -e "CREATE USER 'roundesk'@'localhost' IDENTIFIED BY 'some_pass';"

- name: t3
  shell:
    mysql -uroot -proot -e "GRANT ALL PRIVILEGES ON *.* TO 'roundesk'@'localhost' WITH GRANT OPTION;"

- name: t1
  shell:
    mysql -uroot -proot -e "CREATE USER 'roundesk'@'%' IDENTIFIED BY 'some_pass';"

- name: t1
  shell:
    mysql -uroot -proot -e "GRANT ALL PRIVILEGES ON *.* TO 'roundesk'@'%' WITH GRANT OPTION;"

- name: t1
  shell:
    mysql -uroot -proot -e "FLUSH PRIVILEGES;"

- name: create db asterisk
  shell:
    mysql -uroundesk -psome_pass -e "create database asterisk;"

- name: create db pmr
  shell:
    mysql -uroundesk -psome_pass -e "create database pmr_chat;"

- name: copy database asterisk
  copy:
    src: ../asterisk.sql
    dest: ~/asterisk.sql
    mode: '0644'

- name: copy database pmr_chat
  copy:
    src: ../final_pmr.sql
    dest: ~/final_pmr.sql
    mode: '0644'

- name: import pmr
  shell: mysql -uroundesk -psome_pass pmr_chat <~/final_pmr.sql

- name: import asterisk
  shell: mysql -uroundesk -psome_pass asterisk <~/asterisk.sql
```

FIG. 5.13 – : Role config-db main.yml

La figure ci-dessus représente le fichier principal de rôle « config-db » ce rôle permet de créer un utilisateur MYSQL pour gérer la base de données, puis créer les 2 de base nécessaire et finalement importer les fichiers SQL contenant les données, en utilisant le module « shell ».


```

source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: (remote-exec) warning: Using a password on the command line interface can be insecure.
source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: Still creating... [60s elapsed]
source linux_virtual_machine.license: Creation complete after 60s [id=subscription/6d68329-3940-407c-93ca-8248523e3a/resourcegroups/akidaki-rg/providers/Microsoft.Compute/virtualMachines/akidaki]

Apply complete! Resources: 12 added, 0 changed, 0 destroyed.

Outputs:
ip = "51.103.74.194"
terraformname-6158700~Desktop\source\akidaki\src\

```

FIG. 5.14 – : Le Résultat de terraform Apply

5.2.3 Déploiement Continu

Après avoir mis en place les outils d'environnement, on doit créer le pipeline d'intégration et déploiement continu. Il est illustré à travers une séquence de phases qui représentent le flux de travail du lancement de build jusqu'au déploiement en utilisant Ansible.

Ceci était réalisé à travers des stages de pipeline Gitlab CI, chaque stage vise une fonctionnalité bien précise. La séquence des captures d'écran ci-dessous définit chaque étape réalisée pour atteindre cela, le fichier sera exécuté quand on fait un commit sur la branche Develop.

Dans le pipeline on a seulement deux stages, Build Deploy Le premier est utilisé pour dockeriser l'application et l'autre pour la déployer.

□ Dockerfile

Un Dockerfile est un document texte qui contient toutes les commandes qu'un utilisateur peut appeler sur la ligne de commande pour assembler une image. À l'aide de docker build, les utilisateurs peuvent créer une build automatisée qui exécute successivement plusieurs instructions de ligne de commande. La figure ci-dessous représente le dockerfile créé pour notre projet. Le Dockerfile_{evestercommececi} :

- Choisir l'image de base (dans notre cas lampp qui contient tous les outils pour exécuter un projet Symfony) .
- Spécifier le dossier de travail (Work directory).
- Copier tous les fichiers dans ce dossier.

- Désactiver le mode prod et activer le mode dev de l'application.
- Modifier la configuration des deux bases de données.
- Exécuter le build de l'application en utilisant le `composer.phar`.
- Allouer l'accès au port 80 (le port de l'application).
- Finalement mettre trois commandes qui seront exécutées lors de l'exécution du conteneur (la mise à jour et la validation des bases de données et l'exécution de l'application sur le port 80).

```
FROM vgwdev/lampp
WORKDIR /app
COPY . /app
RUN sed -i -e 's/dev/prod/g' web/app_dev.php
RUN mv -f db_asterisk_pdo_dev.php web/api/db_asterisk_pdo.php
RUN mv -f param_dev.yml app/config/parameters.yml.dist
RUN php composer.phar install
EXPOSE 80
CMD php app/console doctrine:schema:update --force; php app/console doctrine:schema:validate ; php app/console server:run 0.0.0.0:80;
```

FIG. 5.15 – Dockerfile dev

□ Le fichier Ansible playbook

Comme la figure ci-dessous montre, le fichier `playbook` contient les étapes à exécuter afin de lancer le conteneur docker avec notre projet, au début, nous nous connectons au registre de conteneurs gitlab, puis nous supprimons le conteneur précédent s'il existe et exécutons la version la plus récente, enfin nous supprimons les images docker inutilisées.

```
---
- hosts: azure
  become: true
  gather_facts: no
  tasks:

    - name: Log into DockerHub
      docker_login:
        username: safwen@roundesk.net
        password: "{{ pass }}"
        registry: registry.gitlab.com

    - name: run container
      docker_container:
        name: "okidoki"
        image: "{{ Image_tag }}"
        recreate: yes
        ports:
          - 8080:80
```

FIG. 5.16 – Ansible Playbook

□ Gitlab-ci.yml

Le fichier, Gitlab-ci.yml est un fichier YAML que doit être créé la racine de projet. Ce fichier s'exécute automatiquement chaque fois on envoie un commit au serveur. Cela déclenche une notification au « Gitlab-runner », puis il traite la série de tâches qu'on a spécifié. Dans notre cas on a 2 stages, la partie « build » qui va créer une image docker avec le fichier Dockerfile-dev déjà créée et puis la partie « deploy » qui sert à exécuter le fichier ansible playbook déjà mentionné.

```

gitlab-ci.yml
  stages:
    - build
    - deploy
  variables:
    IMAGE_TAG: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA

  docker_image_for_dev:
    stage: build
    image: docker:18

  services:
    - docker:dind

  variables:
    DOCKER_HOST: tcp://docker:2375

  script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN registry.gitlab.com
    - docker build --pull -t "$IMAGE_TAG" -f Dockerfile_dev .
    - docker push "$IMAGE_TAG"

  only:
    - develop

  deploy on dev:
    stage: deploy
    image: bertitodocker/ansible_ubuntu_20.04

  script:
    - touch key.pem
    - mv $sshkey key.pem
    - chmod 400 key.pem
    - ansible-playbook -i ansible/configs ansible/playbook.yaml -e Image_tag="$IMAGE_TAG" -e pass=$dockerpass

  only:
    - develop
  
```

FIG. 5.17 – Gitlab-ci.yaml

5.3 L'environnement Préprod et Prods

Les environnements Préprod et Prod doivent être identiques, pour cela on a décidé de créer cluster AKS (Azure Kubernetes Service) avec deux namespaces différents une pour le préprod et la deuxième pour la prod.

5.3.1 Les variables Terraform

```
variable "resource_group_name" {
  type      = string
  description = "RG name in Azure"
}
variable "resource_group_location" {
  type      = string
  description = "RG location in Azure"
}
variable "cluster_name" {
  type      = string
  description = "cluster name in Azure"
}
variable "dns_prefix" {
  type      = string
  description = "dns_prefix in Azure"
}
variable "node_Count" {
  type      = number
  description = "node count in Azure"
}
variable "VM_Size" {
  type      = string
  description = "VM size in Azure"
}
```

FIG. 5.18 – Fichier Variables.tf

```
resource_group_name      = "okidoki_rg"
resource_group_location  = "westus2"
cluster_name             = "okidoki"
dns_prefix               = "okidoki"
node_Count               = 1
VM_Size                  = "Standard_E2_v3"
```

FIG. 5.19 – Fichier Terraform.tfvars

- `Variables.tf` est utilisé pour définir le type de variables et éventuellement définir une valeur par défaut
- `Terraform.tfvars` contient toutes les valeurs des variables présentées dans le fichier `variables.tf`

5.3.2 Provisionnement du Cluster

Dans cette partie on présente le fichier main.tf de terraform qui est responsable du provisionnement et la création du cluster, on présente chaque block en expliquant son rôle.

```
terraform {
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = ">= 2.26"
    }
  }
}

provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "okidoki_rg" {
  name     = var.resource_group_name
  location = var.resource_group_location
}

resource "azurerm_kubernetes_cluster" "okidoki_aks" {
  name                = var.cluster_name
  location             = azurerm_resource_group.okidoki_rg.location
  resource_group_name = azurerm_resource_group.okidoki_rg.name
  dns_prefix          = var.dns_prefix
  default_node_pool {
    name     = "default"
    node_count = var.node_Count
    vm_size  = var.VM_Size
  }
  identity { type = "SystemAssigned" }
}

resource "null_resource" "kubect1" {
  provisioner "local-exec" {
    command = <<EOF
echo "${azurerm_kubernetes_cluster.okidoki_aks.kube_config_raw}" > ~/config;
bash script.sh
EOF
  }
}
```

FIG. 5.20 – main.tf

- **Provider** : le fournisseur cloud a utilisé
- **azurerm-resource-group** : créer un groupe de ressource si n'existe pas
- **azurerm-kubernetes-cluster** : créer un cluster AKS avec un nom, location dans le groupe de ressource déjà mentionner

- **default-node-pool** : c'est un sous block contient la configuration nécessaire pour un cluster (le nombre des nœuds souhaités, et a taille de VM)
- **null-resource** : permet d'exécuter une commande sur local avec la ressource déjà créée , dans notre cas , d'enregistrer la configuration de AKS cluster dans un fichier config

5.3.3 Mise en place des outils nécessaire

Après l'approvisionnement et quand le cluster est prés on doit installer quelques outils configurer la base de données, on a créé un fichier script.sh permettant d'installer et configurer ces outils. Ce fichier permet de :

- Créer les namespaces "MySQL", "MySQL-pré-prod", "ingress", "cert-manager" et "prometheus".
- Installer mysql-operator dans le namespace MySQL.
- Création d'une instance mysql-cluster () dans le namespace MySQL-pre-prod.
- Importation des bases de données dans mysql-cluster du namespace MySQL-pre-prod.
- Installer cert-manager dans le namespace cert-manager.
- Installer ingress-Nginx dans le namespace ingress.

A) Mise en place de MySQL Opérateur

```
helm repo add presslabs https://presslabs.github.io/charts;  
helm repo update ;  
helm install mysql-operator presslabs/mysql-operator --namespace mysql --create-namespace --kubeconfig-config;  
  
while [[ $(kubectl get pods mysql-operator-0 -n mysql -o 'jsonpath={..status.conditions[?(@.type=="Ready")].status}') != "True" ]]; do echo "waiting for pod" && sleep 5; done
```

FIG. 5.21 – Installation de MySQL Operator

La figure ci-dessus (Voir figure 5.21) présente la partie d'installation de `mysql-operator`, on commence par ajouter le repo de helm chart puis mise à jour de helm repo liste, et enfin installer avec helm le MySQL Operator dans un namespace déjà créé, Finalement on attend la création de pod

B) Installation de MySQL Cluster

```
helm repo add presslabs https://presslabs.github.io/charts;
helm repo update ;
helm install mysql-operator presslabs/mysql-operator --namespace mysql --create-namespace --kubeconfig-config;

while [[ $(kubectl get pods mysql-operator-0 -n mysql -o 'jsonpath={..status.conditions[?(@.type=="Ready")].status}') != "True" ]]; do echo "waiting for pod" && sleep 5; done
```

FIG. 5.22 – Installation de MySQL Cluster

La Figure ci-dessus (Voir figure 5.23) représente la mise en place de MySQL Cluster dans un namespace préprod, on commence par un namespace nommé MySQL-préprod puis appliquer le cluster et le secret de MySQL. Finalement, on attend la création de Cluster avant de continuer vers l'étape suivante.

```
apiVersion: mysql.presslabs.org/v1alpha1
kind: MysqlCluster
metadata:
  name: my-cluster-prod
spec:
  selector:
selector:
  matchLabels:
    app.kubernetes.io/managed-by: mysql.presslabs.org
    app.kubernetes.io/name: mysql
    mysql.presslabs.org/cluster: my-cluster-prod
spec:
  replicas: 2
  secretName: my-secret
```

FIG. 5.23 – mysql-cluster.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  ROOT_PASSWORD: QXplcnR5MTIz
```

FIG. 5.24 – mysql-secret.yaml

C) Création de la base de données

Après la création et mise en place du cluster, on a ajouté un utilisateur aux bases de données, avec tous les privilèges, puis on a créé les bases de données et finalement, on importe les données dans les bases déjà créées.

```
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroot -pAzerty123 -e "CREATE USER 'roundesk'@'localhost' IDENTIFIED BY 'roundesk';"
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroot -pAzerty123 -e "GRANT ALL PRIVILEGES ON . TO 'roundesk'@'localhost';"
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroot -pAzerty123 -e "CREATE USER 'roundesk'@'%' IDENTIFIED BY 'roundesk';"
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroot -pAzerty123 -e "GRANT ALL PRIVILEGES ON . TO 'roundesk'@'%';"
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroot -pAzerty123 -e "FLUSH PRIVILEGES;"
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroundesk -psome_pass -e "create database asterisk;"
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroundesk -psome_pass -e "create database pmr_chat;"
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroundesk -psome_pass pmr_chat < final_pmr.sql
kubectl exec -it my-cluster-mysql-0 -n mysql-pre-prod --kubeconfig-config -- mysql -uroundesk -psome_pass asterisk < asterisk.sql
```

FIG. 5.25 – Création et importation des données

D) Mise en place de la préprod

Après avoir installé les éléments requis, le script va créer le namespace pré-prod, exécuter les services et les besoins dans ce namespace, pour exécuter l'application on a besoin tout d'abord de se connecter au gitlab registry, pour cela Kubernetes offre l'option "imagepullsecrets" qui permet de connecter le déploiement au docker registry en utilisant le type secret comme on a fait dans la figure 35. On a besoin aussi d'un service de type ClusterIP et de contrôleur de type Deployment.

Commençons par le secret :

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-docker
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: ewoJImF1dGhzIjogewoJCSJyZWdpY3RyeS5naXR5YWluY29tI
    jogewoJCQkiYXV0aCI6ICJiVzlvWldSa2FXNWxMbW9oYlIdsc1lVQm
    xjM0J5YVhRdWRHNDZTR0Z0YVd4aE5EQTNMQT09IgoJCX0KCX0KfQoK
```

FIG. 5.26 – registrysecret.yaml

- Le kind représente le type du manifest qu'on va déployer
- Dans metadata on va nommer le secret pour qu'on puisse l'utiliser
- Le bloc type de secret, il existe d'autres types
(<https://docs.openshift.com/dedicated/3/dev-guide/secrets.html#types-of-secrets>).
- Le bloc data contient la valeur du fichier `/.docker/config.json` encodé sur la base 64.

Par la suite on va le service qui va rédiger le trafic vers le contrôleur :

- Le kind représente aussi le type de Manifest, on a besoin d'un service.
- Dans le bloc metadata va nommer le service en utilisant name
- Le bloc spec a plusieurs configurations :

- ⇒ Tout d'abord on va utiliser le bloc selector pour lier le service avec le contrôleur, on doit mettre la même clé et la même valeur dans les deux (le contrôleur et le service).
- ⇒ Par la suite on doit aussi exposer le port de l'application : la clé port c'est le

port qu'on va l'utiliser pour accéder à l'application, le target port c'est le port de l'application qui est exécuté dans le contrôleur/pod

⇒ Finalement le type de service, il existe plusieurs types

(<https://kubernetes.io/fr/docs/concepts/services-networking/service/publishing-services-service-types>) on a besoin de type ClusterIP.

```
kind: Service
apiVersion: v1
metadata:
  name: okidoki-svc
spec:
  selector:
    app: okidoki
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP
```

FIG. 5.27 – okidoki-svc.yaml

Après la préparation des fichiers YAML de la configuration, il ne reste que l'exécution et l'application de ses fichiers en tapant les commandes comment mentionner dans la figure ci-dessous. (Voir figure5.28)

```
kubectl create ns pre-prod --kubeconfig=config;
kubectl apply -f kubefiles/registrysecret.yaml -n pre-prod --kubeconfig=config;
kubectl apply -f kubefiles/okidoki_svc.yaml -n pre-prod --kubeconfig=config;
```

FIG. 5.28 – Application des fichiers services et secret

E) Exposition de l'application déployé

Après l'exécution de ses dernières commandes, notre application est déployée sur le

nouveau namespace pré-prod, mais ne sera pas accessible à l'utilisateur finale, pour cela on a configuré un ingress sécurisé avec un certificat ssl pour et le lié avec notre service déjà crée.

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx ;
helm repo update;
helm install app-ingress ingress-nginx/ingress-nginx --namespace ingress --create-namespace --kubeconfig-config;
while [[ $(kubectl get pods -n ingress -o "jsonpath={.status.conditions[?(@.type=='Ready')].status}") != "True" ]]; do echo "waiting for pod";
helm repo add jetstack https://charts.jetstack.io;
helm repo update;
helm install cert-manager jetstack/cert-manager --namespace cert-manager --set controller.replicaCount=2 --set controller.nodeSelector."beta.kubernetes.io/os=linux" --kubeconfig-config;
kubectl apply -f kubefiles/cluster-issuer.yaml --kubeconfig-config;

sed -i -e 's/<IP>/preprod.$(kubectl get svc app-ingress-ingress-nginx-controller -n ingress --template="{{range .status.loadBalancer.ingress}}{{.ip}}"/'
kubectl apply -f kubefiles/ssl-ingress_pre-prod.yaml -n pre-prod --kubeconfig-config;
echo https://preprod.$(kubectl get svc app-ingress-ingress-nginx-controller -n ingress --template="{{range .status.loadBalancer.ingress}}{{.ip}}"/'
sed -i -e 's/preprod.$(kubectl get svc app-ingress-ingress-nginx-controller -n ingress --template="{{range .status.loadBalancer.ingress}}{{.ip}}"/preprod.<ingress-nginx-IP>.nip.io'
```

FIG. 5.29 – Mise en place d’Nginx Ingress

La figure ci-dessus (Voir figure5.29)représente les étapes de la configuration d’Ingress associé à notre application on commence par ajouter le répo d’ingress avec helm et l’installer dans le namespace désiré . Après l’installation de ingress-nginx il aura une adresse IP public, on va utiliser cette adresse pour exposer l’application pour avoir cette adresse on doit taper “ kubectl get svc app-ingress-ingress-nginx-controller -n ingress --template="range.status.loadBalancer.ingress.ipend" “ et la mettre dans le fichier ssl-ingress-pre-prod.yaml au lieu de <IP>, puisque on utilise un script et on ne peut pas intervenir on va la changer automatiquement en utilisant la commande sed -i -e pour qu’on puisse avoir un domaine comme ceci “ preprod.<ingress-nginx-IP>.nip.io”

Le fichier cluster-issuer.yaml (Voir figure5.30) permet de créer un secret nommé letsencrypt permettant de générer des certificats signés en honorant les demandes de signature de certificat, on doit définir un serveur et un email, aussi le composant (ingress nginx) qui va l’utiliser.


```
apiVersion: cert-manager.io/v1alpha2
kind: ClusterIssuer
metadata:
  name: letsencrypt
  labels:
    name: letsencrypt
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: safwen@roundesk.net
    privateKeySecretRef:
      name: letsencrypt
    solvers:
    - http01:
        ingress:
          class: nginx
```

FIG. 5.30 – Fichier Cluster-issuer.yaml

Le dernier fichier c'est ssl-ingress-pre-prod nous permet de créer un objet de type ingress, il gère l'accès externe aux services dans un cluster, généralement du trafic HTTP.

La configuration se fait sous le bloc spec :

- ✧ Si on veut sécuriser le trafic on doit mettre le bloc tls : sous ce bloc on doit spécifier le nom de domaine et le nom de secret, ce dernier est généré lors de la création du cluster-issuer.
- ✧ Le bloc rules permet de définir des règles, on va mettre une autre fois le nom de domaine, et la configuration qui va le lier avec l'application, sous le bloc backend on va mettre le nom du service et le numéro du port de l'application.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ssl-tls-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt
spec:
  tls:
    - hosts:
        - <IP>
      secretName: ssl-web-cert
  rules:
    - host: <IP>
      http:
        paths:
          - backend:
              serviceName: okidoki-svc
              servicePort: 80
            path: /
```

FIG. 5.31 – ssl-ingress-pre-prod.yaml - avant l'exécution du script

La figure ci-dessous (Voir figure 5.31) est le résultat de commande sed qui va changer <IP> par le nom de domaine qu'on va l'utiliser pour le pré-prod .

On peut tester maintenant si la sécurisation fonctionne ou non, en accédant sur le nom de domaine créé

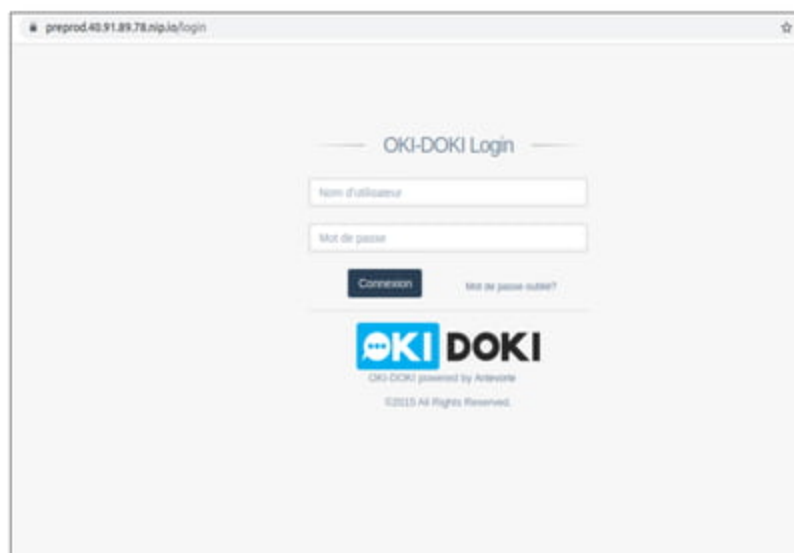


FIG. 5.32 – Test de déploiement

5.3.4 Déploiement Continu

Le déploiement se fait aussi automatiquement après le commit du code sur git, pour cela on a créé un pipeline d'intégration et déploiement continue pour l'environnement pré-prod. Il est illustré à travers une séquence de phases qui représentent le flux de travail du lancement de build jusqu'au déploiement en utilisant Kubectl. Ceci était réalisé à travers des stages de pipeline Gitlab CI, chaque stage vise une fonctionnalité bien précise. La séquence des captures d'écran ci-dessous définit chaque étape réalisée pour atteindre. Cela va être exécuté quand on fait un commit sur la branche Master. Dans le pipeline on a seulement deux stages, Build Deploy

Le premier est utilisé pour dockeriser l'application et l'autre pour la déployer, c'est presque comme la branche develop.

```
docker_image_for_prod:
  stage: build
  image: docker:18
  services:
    - docker:dind
  variables:
    DOCKER_HOST: tcp://docker:2375

  script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN registry.gitlab.com
    - docker build --pull -t "$IMAGE_TAG" -f Dockerfile_prod .
    - docker push "$IMAGE_TAG"

  only:
    - master

deploy on master:
  stage: deploy
  image: dtzar/helm-kubectl

  script:
    - sed -i "s|<image-name>|${IMAGE_TAG}|g" kubefiles/okidoki_deploy.yaml
    - kubectl apply -f kubefiles/okidoki_deploy.yaml --kubeconfig-kubefiles/config --namespace=pre-prod

  only:
    - master
```

FIG. 5.33 – Fichier Gitlab-Ci.yaml

On va faire les même étapes que le build de l'image de l'environnement de test avec quelque changement du Dockerfile.

Le Dockerfile-prod est créé comme ceci :

⇒ A la ligne 4 on va créer une variable d'environnement DB-HOST car on va utiliser

la même image pour deux environnements(prod et pré-prod) avec deux bases de données différents

- ⇒ Aux les lignes 6 et 7 on a changé la configuration des deux bases de données de l'environnement pré-prod.
- ⇒ La ligne 10 et 11 va changer l' hôte des bases de données dans les deux fichiers de configuration des bases de données.
- ⇒ Le reste est similaire à la configuration du dockerfile d'environnement test.

```
FROM vgudev/lamp
WORKDIR /app
COPY . /app
RUN sed -i -e 's/dev/prod/g' web/app_dev.php
RUN mv -f db_asterisk_pdo_prod.php web/api/db_asterisk_pdo.php
RUN mv -f param_prod.yml app/config/parameters.yml.dist
RUN php composer.phar install

EXPOSE 80

CMD php app/console doctrine:schema:update --force; php app/console doctrine:schema:validate;php app/console server:run 0.0.0.0:80
```

FIG. 5.34 – : Dockerfile-prod

On a le contrôleur qui va hoster notre pods :

- ⇒ Le kind représente le type de manifest, il existe plusieurs types de contrôleur(<https://kubernetes.io/fr/docs/concepts/workloads/controllers/>) on besoin d'un contrôleur de type Deployment.
- ⇒ Dans metadata on va nommer le Contrôleur en utilisant name
- ⇒ Le bloc spec a plusieurs configuration :
 - Replicas : définir le nombre des pods à exécuter, on a besoin seulement d'un pod.
 - Selector.matchlabels et template.metadata.labels : définir un label pour relier le contrôleur et le service.
 - le bloc spec est le même lorsqu'on définit un pod, cela signifi qu'on va définir le pod ici,

- Containers est le bloc où on va définir le nom de pod, l'image (dans cette partie on va changer le tag manuellement avant de lancer tous, on va mettre le dernier tag de la dernière image construit) et les ports de l'application (le targetport dans le service et le port exposé dans le dockerfile).
- Imagepullsecrets est le bloc où on relie le contrôleur avec le secret qu'on a créé précédemment pour pouvoir puller l'image Docker).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: okidoki-dep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: okidoki
  template:
    metadata:
      labels:
        app: okidoki
    spec:
      containers:
        - name: okidoki
          image: <image-name>
          ports:
            - containerPort: 80
      imagePullSecrets:
        - name: secret-docker
```

FIG. 5.35 – Oki-Doki-Deployment.yaml

Après la livraison de l'image Docker sur Gitlab registry, il est temps de déployer l'application.

Dans la partie de déploiement on a choisi l'image docker "dtzar/helm-kubectrl" sur laquelle on trouve Kubectl installé pour éviter son installation pour gagner plus de temps, les étapes suivies sont comme ceci :

On va faire un petit changement sur le fichier okidoki-deploy.yaml ; on va mettre <image-name> dans le bloc images pour qu'on puisse changer le nom automatiquement avec la commande sed par le nom du nouvelle image.

la dernière commande c'est Kubectl apply qui va appliquer le fichier de déploiement modifié pour faire le redéploiement,, l'option f spécifie le fichier voulu, on va appliquer cet objet sur le namespace pré-prod en spécifiant le cluster avec l'option --kubeconfig.

NB : le dossier kubefiles qui contient les deux fichiers config et okidoki-deploy.yaml se trouve sous le répertoire du projet okidoki.

```
deploy on master:
  stage: deploy
  image: dtzar/helm-kubectl

  script:
    - sed -i "s|<image-name>|${IMAGE_TAG}|g" kubefiles/okidoki_deploy.yaml
    - kubectl apply -f kubefiles/okidoki_deploy.yaml --kubeconfig=kubefiles/config --namespace=pre-prod

  only:
    - master
```

FIG. 5.36 – gitlab-ci.yml - stage de déploiement

5.3.5 Exécution de pipeline :

Après pusher le code vers gitlab , En accédant a [Gitlab.com/nom-d'utilisateur/nom-du-repo/pipelines](https://gitlab.com/nom-d'utilisateur/nom-du-repo/pipelines) nous pouvons trouver la liste des pipelines, en choisissant le pipeline du notre commit, nous pouvons suivre l'avancement du pipeline.

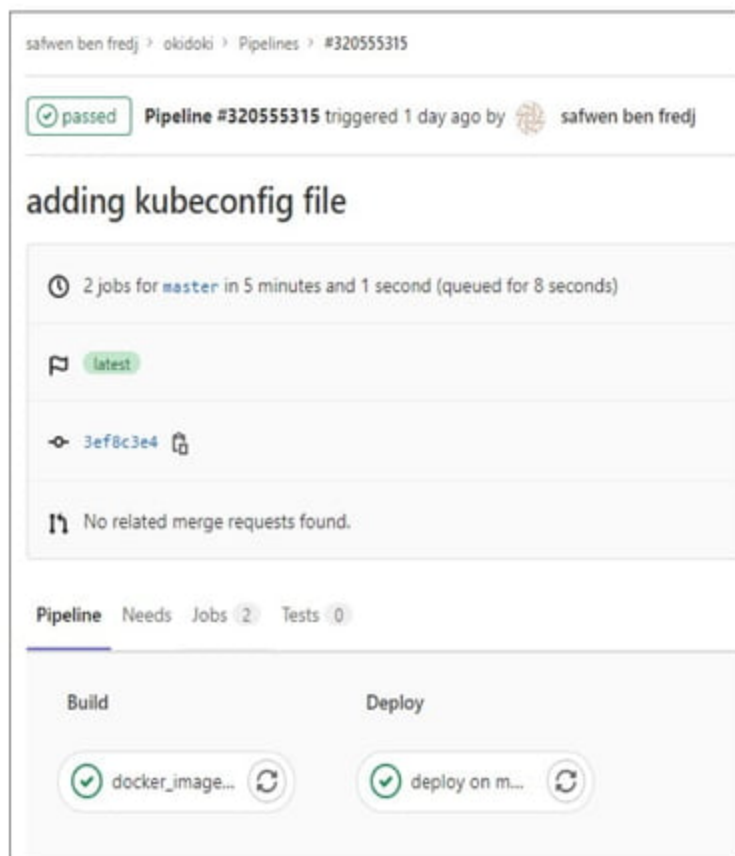


FIG. 5.37 – Avancement de gitalb pipeline

5.4 L'environnement production

Après avoir déployé une version préprod avec un fonctionnement correcte , on a fait le même process dans un namespace différent dans le cluster déjà créé dans le provisionnement dans la partie préprod.

5.4.1 Mise en place des outils

Dans cette partie on va faire les memes etapes faites dans la mise en place des outils mentioné dans la partie précédente (mise en place de l'environnement préprod) , et déployé notre application .

On va faire la configuration nécessaire d'ingress-nginx,cert-manager,et mysql-operator qui

sont déjà installé dans le cluster .

On commence par installer MySQL cluster qui sera dédié pour cet environnement , l'installation se fait comme ceci :

- Crée un secret qui contient le mot de passe du root en appliquant le fichier de mysql-secret avec la commande « Kubectl create -f kubefiles/mysql-secret.yaml -n mysql-prod --kubeconfig=config ».

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  ROOT_PASSWORD: QXplcnR5MTIz
```

FIG. 5.38 – MySQL-secret.yaml

- Après avoir installé le secret on va exécuter le fichier mysql-cluster-prod.yaml sur le namespace MySQL .

```

apiVersion: mysql.presslabs.org/v1alpha1
kind: MysqlCluster
metadata:
  name: my-cluster-prod
spec:
  selector:
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: mysql.presslabs.org
      app.kubernetes.io/name: mysql
      mysql.presslabs.org/cluster: my-cluster-prod
spec:
  replicas: 2
  secretName: my-secret

```

FIG. 5.39 – mysql-cluster-prod.yaml

- Pour vérifier la création de cluster on tape la commande « `kubectl get all -n mysql-prod --kubeconfig=config` » .

```

safouene@safouene-GL503VD:~/Downloads/k8s-2$ kubectl get pods -n mysql-prod --kubeconfig=config
NAME                READY   STATUS    RESTARTS   AGE
my-cluster-mysql-0   4/4     Running   0           2m47s
my-cluster-mysql-1   4/4     Running   0           92s

```

FIG. 5.40 – vérification de la mise en place de mysql-cluster pod

Après la préparation de MySQL, il reste que d'importer les deux bases de données, pour faire cela il faut exécuter quelques commandes dans le pod désiré comme mentionner dans la figure ci-dessous (Voir figure5.46)


```

safouene@safoene-GL503VD:~/Downloads/10-1$ kubectl exec -it my-cluster-mysql-0 -n mysql-prod --kubeconfig=config --mysql -uroot -pAerty123 -e "CREATE USER 'roundek'@'localhost' IDENTIFIED BY 'some_pass';"
Defaulting container name to mysql.
Use 'kubectl describe pod/my-cluster-mysql-0 -n mysql-prod' to see all of the containers in this pod.
mysql: [Warning] Using a password on the command line interface can be insecure.
safouene@safoene-GL503VD:~/Downloads/10-1$ kubectl exec -it my-cluster-mysql-0 -n mysql-prod --kubeconfig=config --mysql -uroot -pAerty123 -e "GRANT ALL PRIVILEGES ON *.* TO 'roundek'@'localhost' WITH GRANT OPTION;"
Defaulting container name to mysql.
Use 'kubectl describe pod/my-cluster-mysql-0 -n mysql-prod' to see all of the containers in this pod.
mysql: [Warning] Using a password on the command line interface can be insecure.
safouene@safoene-GL503VD:~/Downloads/10-1$ kubectl exec -it my-cluster-mysql-0 -n mysql-prod --kubeconfig=config --mysql -uroot -pAerty123 -e "CREATE USER 'roundek'@'%' IDENTIFIED BY 'some_pass';"
Defaulting container name to mysql.
Use 'kubectl describe pod/my-cluster-mysql-0 -n mysql-prod' to see all of the containers in this pod.
mysql: [Warning] Using a password on the command line interface can be insecure.
safouene@safoene-GL503VD:~/Downloads/10-1$ kubectl exec -it my-cluster-mysql-0 -n mysql-prod --kubeconfig=config --mysql -uroot -pAerty123 -e "GRANT ALL PRIVILEGES ON *.* TO 'roundek'@'%' WITH GRANT OPTION;"
Defaulting container name to mysql.
Use 'kubectl describe pod/my-cluster-mysql-0 -n mysql-prod' to see all of the containers in this pod.
mysql: [Warning] Using a password on the command line interface can be insecure.
safouene@safoene-GL503VD:~/Downloads/10-1$ kubectl exec -it my-cluster-mysql-0 -n mysql-prod --kubeconfig=config --mysql -uroot -pAerty123 -e "FLUSH PRIVILEGES;"
Defaulting container name to mysql.
Use 'kubectl describe pod/my-cluster-mysql-0 -n mysql-prod' to see all of the containers in this pod.
mysql: [Warning] Using a password on the command line interface can be insecure.
safouene@safoene-GL503VD:~/Downloads/10-1$ kubectl exec -it my-cluster-mysql-0 -n mysql-prod --kubeconfig=config --mysql -uroot -pAerty123 -e "create database asterisk;"
Defaulting container name to mysql.
Use 'kubectl describe pod/my-cluster-mysql-0 -n mysql-prod' to see all of the containers in this pod.
mysql: [Warning] Using a password on the command line interface can be insecure.
safouene@safoene-GL503VD:~/Downloads/10-1$ kubectl exec -it my-cluster-mysql-0 -n mysql-prod --kubeconfig=config --mysql -uroot -pAerty123 -e "create database pm_chat;"
Defaulting container name to mysql.
Use 'kubectl describe pod/my-cluster-mysql-0 -n mysql-prod' to see all of the containers in this pod.
mysql: [Warning] Using a password on the command line interface can be insecure.

```

FIG. 5.41 – Mise en place de bases de données prod

5.4.2 Déploiement sur le cluster

On va refaire l'étape du déploiement sur l'environnement pré-prod mais manuellement avec une vérification pour chaque étape si elle fonctionne correctement ou non, nous devons créer le namespace prod en tapant `kubectl create ns prod --kubeconfig=config`.

```

safouene@safoene-GL503VD:~/kube$ kubectl create ns mysql-prod --kubeconfig=config ;
namespace/mysql-prod created

```

FIG. 5.42 – l'environnement prod - vérification de la création du namespace prod

Par la suite on va exécuter l'application dans ce namespace, pour exécuter l'application, il faut tout d'abord appliquer le fichier `registry-secret.yaml`

```

apiVersion: v1
kind: Secret
metadata:
  name: secret-docker
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: ewoJImFldGhZlJogewoJCS3yZWdpc3RyeS5naX

```

FIG. 5.43 – registry-secret.yaml

Par la suite on prépare notre service qui sert à rédiger le trafic vers le contrôleur


```
kind: Service
apiVersion: v1
metadata:
  name: okidoki-svc
spec:
  selector:
    app: okidoki
  ports:
    - port: 80
      targetPort: 80
  type: ClusterIP
```

FIG. 5.44 – okidoki-svc.yaml

Nous devons accéder au container registry du gitlab pour avoir le tag de l'image et le changer manuellement. Le bloc ENV sous spec.containers va override l'hôte des bases de données.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: okidoki-dep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: okidoki
  template:
    metadata:
      labels:
        app: okidoki
    spec:
      containers:
        - name: okidoki
          image: registry.gitlab.com/safwen/bf/okidoki:858ef7ce
          env:
            - name: DB_HOST
              value: my-cluster-mysql-0.mysql.mysql-prod.svc.cluster.local
          ports:
            - containerPort: 80
          imagePullSecrets:
            - name: secret-docker
```

FIG. 5.45 – Okidoki-deploy.yaml

On doit aussi sécuriser notre application, le cluster-issuer est déjà installé, il reste que le ingress pour l'environnement prod.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: okidoki-dep
spec:
  replicas: 1
  selector:
    matchLabels:
      app: okidoki
  template:
    metadata:
      labels:
        app: okidoki
    spec:
      containers:
        - name: okidoki
          image: registry.gitlab.com/safwen.bf/okidoki:858ef7ce
          env:
            - name: DB_HOST
              value: my-cluster-mysql-0.mysql.mysql-prod.svc.cluster.local
          ports:
            - containerPort: 80
          imagePullSecrets:
            - name: secret-docker
```

FIG. 5.46 – ssl-ingress-prod.yaml

Maintenant l'application est déployée est prêt à être utiliser et tester, la figure ci-dessous (Voir figure5.47) montre le résultat

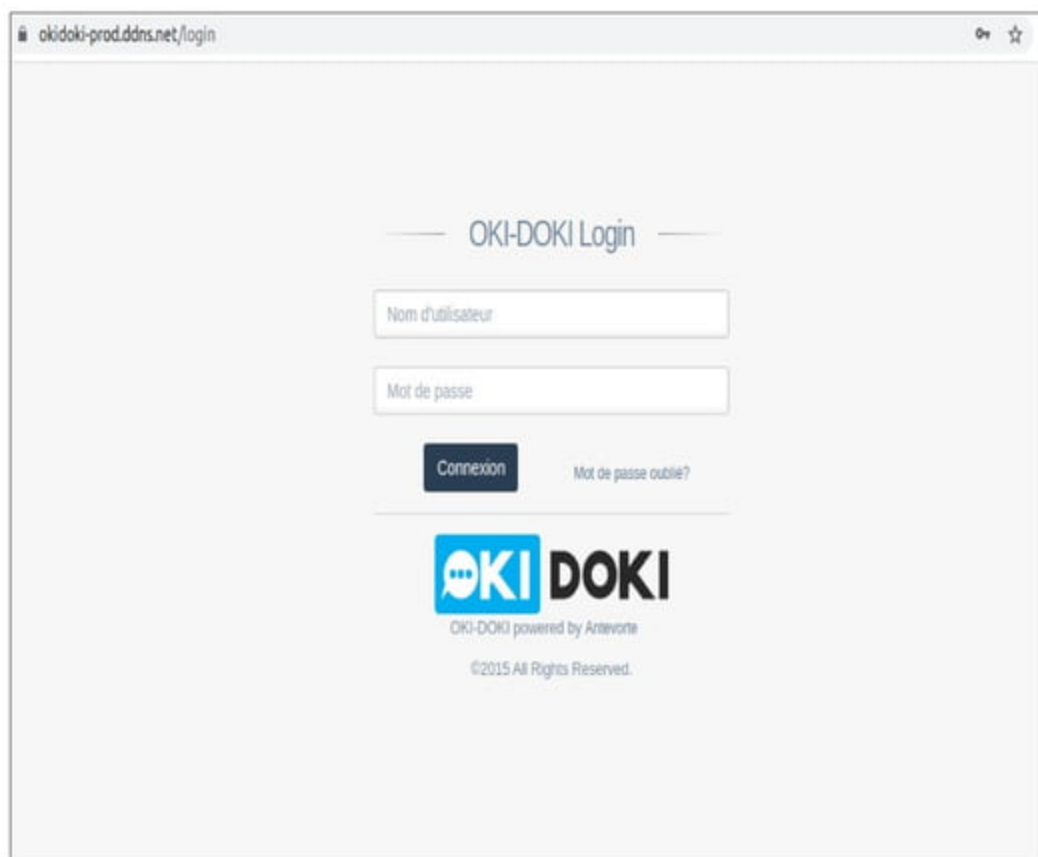


FIG. 5.47 – Interface de l'application déployée

5.5 Monitoring

5.5.1 Installation des outils

Pour faciliter l'installation et pour gagner du temps nous avons utilisé HELM pour faire l'installation de Prometheus et Grafana. Heureusement, nous avons trouvé une charte prête qui répond à ce besoin. Nous devons tout d'abord ajouter cette charte, faire une mise à jour du HELM repository et l'installer, la figure ci-dessous (Voir figure 5.48) représente la liste des instructions dans le fichier script, cela veut dire qu'on l'installe automatiquement

```

10
11 helm repo add stable/prometheus-operator https://prometheus-operator.github.io/prometheus-operator/charts/
12 helm repo update;
13 helm install prometheus stable/prometheus-operator --namespace monitoring --create-namespace
14
75 |

```

FIG. 5.48 – Script de l'environnement pré-prod - installation du Prometheus

Après l'installation, nous allons lister les composants créés par HELM. Pour le faire, nous devons taper "kubectl get all -n monitoring --kubeconfig=config"

```

jfovenagafovenaga-CL503V0:~/Desktop/azure_okt/azure/k8s-15$ kubectl get all -n monitoring --kubeconfig=config
NAME                                READY    STATUS    RESTARTS   AGE
pod/alertmanager-prometheus-prometheus-oper-alertmanager-0  2/2      Running   0           83m
pod/prometheus-grafana-7c78857f5c-74zds  2/2      Running   0           83m
pod/prometheus-kube-state-metrics-95d956569-tgrfn  1/1      Running   0           83m
pod/prometheus-prometheus-node-exporter-tpd8t  1/1      Running   0           83m
pod/prometheus-prometheus-oper-operator-dd9c4bd9f-5jgqk  2/2      Running   0           83m
pod/prometheus-prometheus-prometheus-oper-prometheus  3/3      Running   1           83m

NAME                                TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
service/alertmanager-operated       ClusterIP  None          <none>         9093/TCP,9094/TCP,9094/UDP  83m
service/prometheus-grafana          ClusterIP  10.0.70.223   <none>         80/TCP           83m
service/prometheus-kube-state-metrics  ClusterIP  10.0.70.88    <none>         8080/TCP         83m
service/prometheus-operated         ClusterIP  None          <none>         9090/TCP         83m
service/prometheus-prometheus-node-exporter  ClusterIP  10.0.159.46   <none>         9100/TCP         83m
service/prometheus-prometheus-oper-alertmanager  ClusterIP  10.0.138.71   <none>         9093/TCP         83m
service/prometheus-prometheus-oper-operator  ClusterIP  10.0.255.127   <none>         8080/TCP,443/TCP  83m
service/prometheus-prometheus-oper-prometheus  ClusterIP  10.0.186.37   <none>         9090/TCP         83m

NAME                                DESIRED    CURRENT    READY    UP-TO-DATE    AVAILABLE    NODE SELECTOR    AGE
daemonset.apps/prometheus-prometheus-node-exporter  1           1           1           1           1           <none>           83m

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/prometheus-grafana  1/1      1           1           83m
deployment.apps/prometheus-kube-state-metrics  1/1      1           1           83m
deployment.apps/prometheus-prometheus-oper-operator  1/1      1           1           83m

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/prometheus-grafana-7c78857f5c  1           1           1           83m
replicaset.apps/prometheus-kube-state-metrics-95d956569  1           1           1           83m
replicaset.apps/prometheus-prometheus-oper-operator-dd9c4bd9f  1           1           1           83m

NAME                                READY    AGE
statefulset.apps/alertmanager-prometheus-prometheus-oper-alertmanager  1/1      83m
statefulset.apps/prometheus-prometheus-prometheus-oper-prometheus  1/1      83m
jfovenagafovenaga-CL503V0:~/Desktop/azure_okt/azure/k8s-15$

```

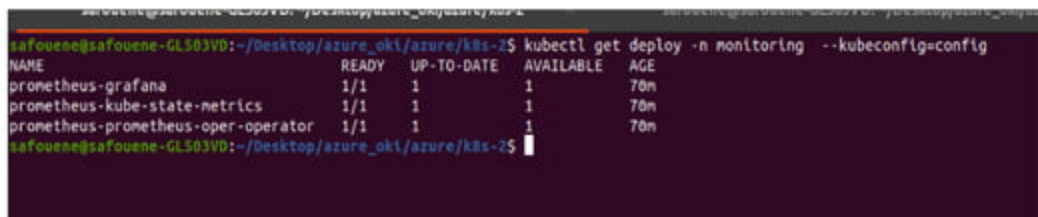
FIG. 5.49 – Pods du monitoring

(Voir figure 5.49) au-dessus liste les composants créés par HELM.

- Le pod alertmanager-prometheus-prometheus-oper-alertmanager-0 contient l'instance de Alert manager qui s'occupe de regroupement et de l'acheminement des alertes de prometheus vers les intégrations de réception correctes, comme le courrier électronique..
- Le pod pod/prometheus-prometheus-prometheus-oper-prometheus-0 contient l'instance de prometheus operator que nous allons l'utiliser pour Puller les métriques.
- Le pod pod/prometheus-grafana-7c78857f5c contient l'instance de Grafana que nous allons l'utiliser pour afficher les métriques .

- Le pod `prometheus-prometheus-node-exporter-46qln` contient l'instance de `node-exporter` qui est l'exportateur Prometheus pour les métriques du matériel et du système d'exploitation est exposé.
- Le pod `prometheus-prometheus-node-exporter-46qln` contient l'instance de `node-exporter` qui écoute le serveur API Kubernetes et génère des métriques sur l'état des objets.

Pour pouvoir utiliser ces Pods, nous avons besoin des contrôleurs, (Voir figure5.50) ci-dessous contient la liste des Deployments qui va hôter ces pods.



```
safovene@safovene-GL503VD:~/Desktop/azure_okt/azure/k8s-2$ kubectl get deploy -n monitoring --kubeconfig=config
NAME                                READY    UP-TO-DATE    AVAILABLE   AGE
prometheus-grafana                  1/1      1              1           70m
prometheus-kube-state-metrics       1/1      1              1           70m
prometheus-prometheus-operator      1/1      1              1           70m
prometheus-prometheus-node-exporter 1/1      1              1           70m
safovene@safovene-GL503VD:~/Desktop/azure_okt/azure/k8s-2$
```

FIG. 5.50 – Deployments du monitoring

Pour pouvoir utiliser ces Déployments, nous avons besoin de les exposer en utilisant des services, (Voir figure5.51) ci-dessous contient la liste des services.

FIG. 5.51 – Services du monitoring

Nous remarquons que tous les services sont du type `ClusterIP`. Pour pouvoir accéder à ces ressources il faut :

- Utiliser le service `LoadBalancer`.
- Utiliser `Ingress`.
- Utiliser `Kubectl port-forward` transfère le trafic vers un port local.

Pour des raisons de confidentialité nous allons utiliser la troisième option "Port-forward" pour exposer ces deux outils.

```
afvenegas@fvenegs-GL502VD:~/Desktop/azure_aki/azure/aks-1$ kubectl port-forward -n monitoring prometheus-prometheus-prometheus-oper-prometheus-0 9090
Forwarding from 127.0.0.1:9090 -> 9090
Forwarding from [::]:9090 -> 9090
```

FIG. 5.52 – Port-forward du prometheus

```
afvenegas@fvenegs-GL502VD:~/Desktop/azure_aki/azure/aks-1$ kubectl port-forward -n monitoring prometheus-grafana-7c78857f5c-74z6s 3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::]:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```

FIG. 5.53 – Port-forward du Grafana

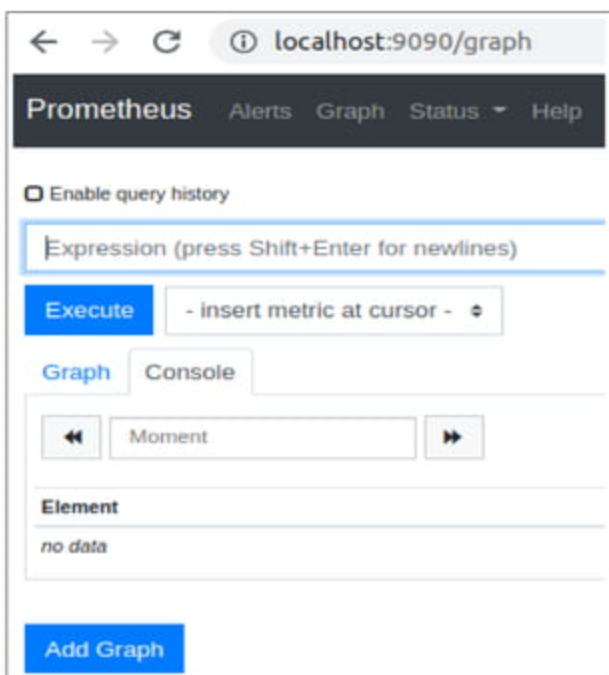


FIG. 5.54 – Interface du Prometheus

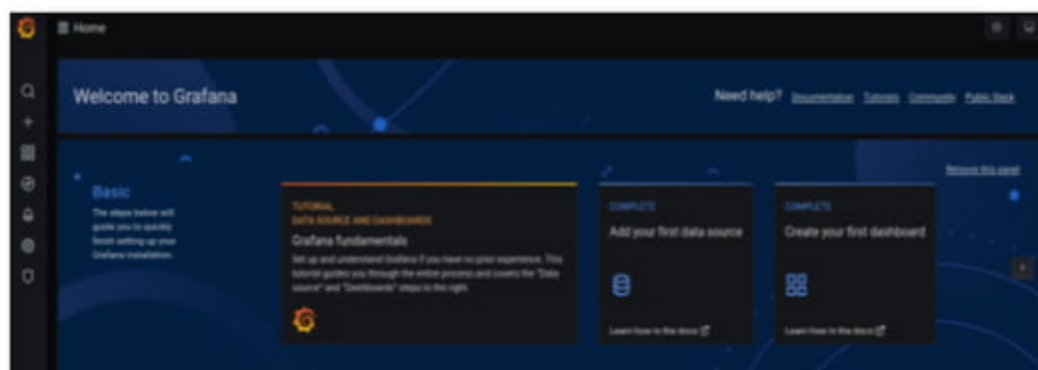


FIG. 5.55 – Interface du Grafana

La communication entre les deux se fait comme ceci, il faut appuyer sur Configuration Data-Sources Add Data-Source Prometheus, et le reste est comme (la figure 5.56) nous montre, nous devons le nommer, ajouter l'URL <http://prometheus-prometheus-oper-prometheus:9090/> et sauvegarder

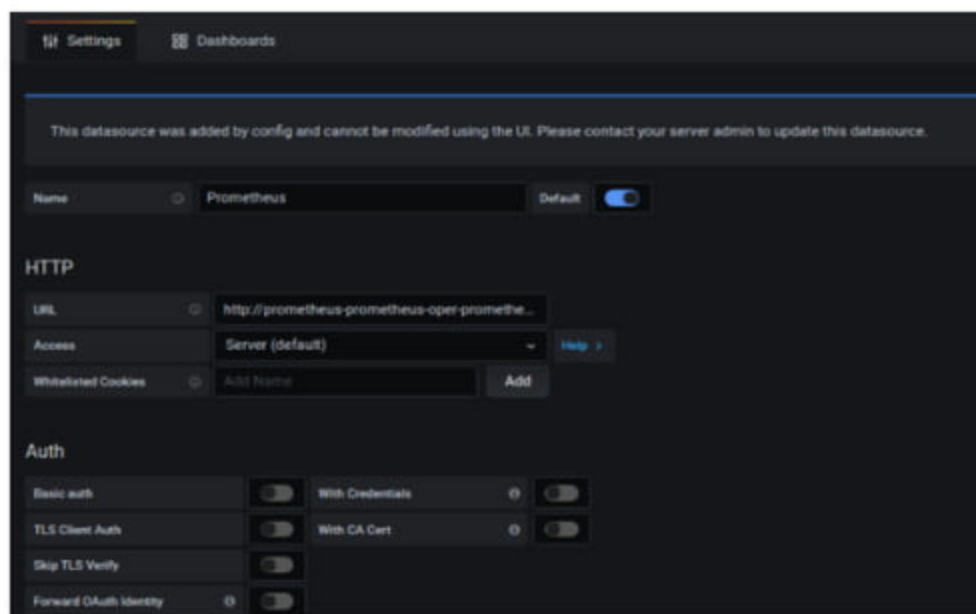


FIG. 5.56 – Configuration du Grafana

5.5.2 Installation des outils

Après avoir préparé les deux composants, nous pouvons maintenant les utiliser pour afficher les métriques désirées, par défaut, Kubernetes envoie des métriques(figure5.57), nous pouvons les trouver dans Dashboard Manage.

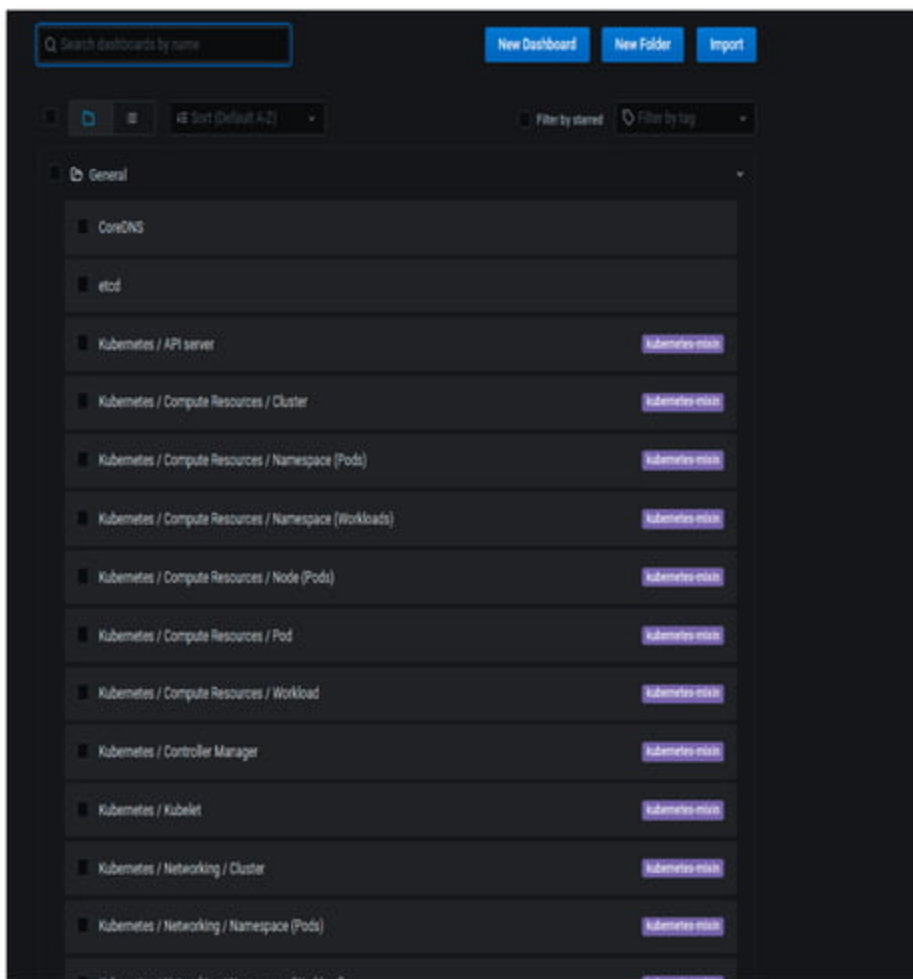


FIG. 5.57 – Liste des métriques à afficher

Il existe deux types de monitoring, monitoring de performances d'application et monitoring de l'infrastructure(Networking, ressources CPU, Mémoire...), nous allons implémenter un ensemble de visualisations qui garantit un aperçu général sur notre infrastructure et notre Cluster Kubernetes. Commençons par le Cluster lui même, (la figure5.58)

représente l'utilisation du CPU et du Mémoire par le cluster



FIG. 5.58 – Utilisation de CPU et de Mémoire par le cluster

chaque visualisation a un rôle bien déterminé :

- 1-4. L'utilisation totale du CPU/mémoire
- 2-5. L'engagement des demandes de CPU/mémoire : Le rapport entre les demandes de CPU/mémoire pour tous les pods fonctionnant sur le cluster et le total des CPU/mémoire disponibles. Nous pouvons également calculer l'engagement de demande au niveau du cluster en comparant les demandes de CPU/mémoire au niveau du cluster à la capacité totale du cluster.
- 3-6. L'engagement de limite de CPU/mémoire : le rapport entre les limites de CPU/mémoire pour tous les pods fonctionnant sur un nœud et le total de CPU/mémoire disponible sur ce nœud. Les engagements de limites au niveau du cluster peuvent être calculés en comparant les limites totales de CPU/mémoire à la capacité du cluster.

Nous pouvons aussi visualiser chaque composant, dans la partie qui suit nous allons se concentrer sur l'utilisation des ressources pour les applications de l'environnement prod et pré-prod et les comparer. (la figure 5.59) montre le charge de CPU du déploiement de l'application Okidoki sur l'environnement prod.



FIG. 5.59 – Charge CPU de l'environnement prod

Par contre, l'application dans l'environnement pré-prod consomme moins de ressources que l'application de l'environnement de prod comme (la figure 5.61) le montre.



FIG. 5.60 – Charge CPU de l'environnement pré-prod

En comparant les deux diagrammes [react] nous pouvons constater mieux la différence. Les clients utilisent l'application à partir de 9h du matin, c'est pour cela que nous observons que l'utilisation du CPU de l'environnement prod est importante que l'utilisation de l'environnement de prod.

A partir de 9h55 du matin, l'utilisation de CPU commence à augmenter progressivement à cause de l'augmentation du nombre d'utilisateurs.



FIG. 5.61 – Charge CPU - l'environnement prod vs l'environnement pré-prod

5.6 Conclusion

Durant ce dernier chapitre, nous avons détaillé notre travail à l'aide des captures d'écran qui présente les différentes étapes de réalisation du projet.

Conclusion Générale

Ce travail fait partie du projet de fin d'études, il a été réalisé au sein de la société Roundesk Technologies. Ce projet est composé de trois étapes, la première consiste à la mise en place de trois environnements, un pour le test, un autre pour la pré-production et le dernier sera dédié pour la production, la deuxième étape se base sur la création d'une chaîne d'intégration, de livraison et de déploiement continue, la dernière étape consiste à la mise en place d'un outil de monitoring.

Pour réaliser ce travail, on a d'abord présenté le cadre général où on a exposé notre étude et critique de l'existant afin de détailler notre solution proposée ainsi que la méthodologie adoptée durant notre travail.

Puis, on va à l'étude préliminaire qu'on a permis de définir les concepts clés du projet, de comparer les différentes solutions techniques pour résumer nos choix. Ensuite, on n'a intéressés à la spécification et à l'analyse, un chapitre qui a fait l'objet de l'identification des interférons, de l'analyse des exigences et de leur représentation.

Enfin, nous on a passés à la partie réalisation qui a porté sur la présentation de l'environnement de travail, les tâches et les étapes réalisées durant le projet à travers des interfaces graphiques.

Grâce à cette expérience, on a pu approfondir nos connaissances en docker, kubernetes, ansible et Terraform et aussi découvrir de nouvelles technologies telles que Azure cloud, Gitlab CI, Prometheus.

En outre, ce projet on a également bénéficié d'un point de vue personnel, il a permis d'appréhender le travail au sein d'une Startup et dans une hiérarchie professionnelle. Le travail de groupe, la gestion de projet de manière méthodique et la répartition des tâches sont des qualités que nous avons acquises grâce à ce stage.

On tient compte d'adapter encore plus de projet développé avec d'autres langages à l'approche DevOps ainsi d'utiliser Sélénium comme outils de test automatique.

Webographie

- [1] URL : <https://harness.io/blog/devops/%20what-is-ci-cd/?fbclid=IwAR221STx6jGBYq1IjlQerkZg8yjlU-iXA4e4X03unPb1WXFrHyDYGdZZgaY/>.
- [2] URL : <https://harness.io/blog/devops/%20what-is-ci-cd/?fbclid=IwAR2sIcep24xqpymGvLKaNy0wunZCd3QGZZZqHX3dDjKpFtfrWfa3Bv9eTo/>.
- [3] URL : <https://www.redhat.com/fr/topics/virtualization>.
- [4] URL : <https://www.tutorialspoint.com/git/index.htm/>.
- [5] URL : https://www.tutorialspoint.com/gitlab/gitlab_ci_introduction.htm.
- [6] URL : <https://www.quora.com/What-is-Docker-in-DevOps#%3A-%3Atext%3DDocker%2C%20a%20container%20management%20tool%2C%20is%20used%20in%20DevOps%20to%2Coverheads%20and%20cuts%20operational%20costs/>.
- [7] URL : <https://indico.mathrice.fr/event/123/session/4/material/0/0.pdf/>.
- [8] URL : <https://www.redhat.com/en/topics/containers/what-is-kubernetes/>.
- [9] URL : <https://www.jesuisundev.com/comprendre-terraform-en-5-minutes/>.
- [10] URL : <https://www.redhat.com/sysadmin/introduction-prometheus-metrics-and-performance-monitoring>.
- [11] URL : <https://www.educba.com/grafana-vs-kibana/>.
- [12] URL : <https://blog.codota.com/svn-vs-git/>.
- [13] URL : <https://upcloud.com/community/stories/%20docker-swarm-vs-kubernetes-comparison-of-thetwo->.
- [14] URL : <https://www.educba.com/ansible-vs-puppet-vs-chef/>.

- [15] URL : <https://www.educba.com/grafana-vs-kibana/>.
- [16] URL : <https://www.katalon.com/resources-center/%20blog/ci-cd-tools/>.
- [17] URL : <https://devblogs.microsoft.com/devops/%20arm-templates-or-hashicorp-terraform-what-should-i-use/>.