

République Algérienne Démocratique et Populaire
Ministre de L'Enseignement Supérieur et de la Recherche Scientifique



UNIVERSITÉ ABDERRAHMANE MIRA BEJAIA
Département d'Informatique
Faculté des sciences Exactes
Spécialité Génie Logiciel

Réaliser par

ADRAR NABILLA

THÈME

Déploiement cloud des algorithmes de machine learning

Devant un jury composé de :
Présidente: Dr. ALOUI Soraya
Examineur: Dr. MOUKATFI Mouhand

Supervisé par:
Encadreur: Dr. BELAID Ahror
Co-encadreur : Mr. AKILAL Abdellah

Remerciements

Je tiens à exprimer ma gratitude à M. Abdellah Akilal et au Dr BELAID Ahror pour leurs soutiens enthousiastes lors de la réalisation de ce projet ainsi que pour leurs conseils et recommandations qui m'ont aidé à mener à bien cette recherche.

Je remercie tous les membres sincères de ma famille, mes parents, mes sœurs et mon frère, qui m'ont soutenu et encouragé tout au long de ma vie, ainsi que pour leur aide précieuse, leur patience et leur soutien indéfectible.

Je tiens à remercier chacun des membres du jury pour l'intérêt qu'ils ont porté à notre travail en acceptant de l'examiner et de l'enrichir de leurs suggestions. Merci à tous les professeurs et membres du Département d'Informatique de l'Université ABDERRAHMANE MIRA de Béjaïa.

Enfin, je remercie mes amis Imene, Fatima, Lyes, Ferhat, Nesrine, Hichem et Fouad d'être toujours à mes côtés. Leur soutien inconditionnel et leurs encouragements ont été d'une grande aide.

Dédicaces

JE DÉDIE CE TRAVAIL À :
MES PARENTS
MES CHERS SOEURS: HAKIMA, SABRINA, FOUZIA, KARIMA ET SAMIA
MON PETIT FRÈRE SOFIANE
MA GRAND-MÈRE QUE SON ÂME REPOSE EN PAIX
TOUS LES MEMBRES DE MA FAMILLE
MES AMIS : IMENE, FATIMA, NESRINE, LYES ET FERHAT
MES TANTES, MES COUSINES ET TATA SAMIRA
HICHEM ET FOUAD AVEC QUI J'AI PARTAGÉ LE FRUIT DE CE TRAVAIL.
TOUTES LES PERSONNES QUI M'ONT OFFERT LEUR AIDE.

Contents

Remerciements	i
Dédicaces	ii
Content	iv
List of Figures	vi
List of Symbols, Abbreviations	vii
Introduction Générale	1
1 Développement et déploiement	2
1.1 Introduction	2
1.2 Développement logiciel	2
1.3 Concept du DevOps:	3
1.4 Déploiement d'applications	5
1.5 Type de déploiement	6
1.6 MLOPS	6
1.6.1 Pourquoi avons-nous besoin de MLOps ?	7
1.7 Stratégie de déploiement	8
1.8 Conclusion	11
2 Machine Learning	12
2.1 Introduction	12
2.2 Machine Learning	12
2.3 Approche et contexte	14
2.4 Déploiement et releases	15
2.5 Services ou microservices	15
2.6 Études des cas	16
2.6.1 Scénario:	16
2.6.2 Secteurs exploitant la Reconnaissance facial	16
2.6.3 Besoins liée au déploiement	17
2.6.4 solution proposée	18
2.7 Conclusion	18
3 Cloud computing	19
3.1 Introduction	19
3.2 Cloud Computing	19
3.3 Modèles de services	19
3.3.1 IaaS:	19

3.3.2	PaaS:	19
3.3.3	SaaS:	20
3.4	Avantages du cloud computing	20
3.5	Virtualisation	21
3.6	Machine virtuel et conteneur	21
3.6.1	Machine virtuel (VM)	21
3.6.2	Conteneurisation	21
3.6.3	Docker	21
3.6.4	Composants Docker	22
3.6.5	Architecture docker	23
3.7	Types de virtualisation	23
3.7.1	La virtualisation de systèmes d'exploitation:	23
3.7.2	La virtualisation d'application:	24
3.8	Kubernetes (orchestration)	25
3.8.1	Définition	25
3.8.2	Fonctionnalités	25
3.8.3	Architecture	25
3.9	Pods et services	27
3.9.1	Pods et nodes	27
3.9.2	Services	28
3.10	ML et kubernetes	31
3.10.1	Kubeflow	31
3.11	Conclusion	32
4	Déploiement local	33
4.1	Introduction	33
4.2	Déploiement avec un seul nœud (Minikube)	33
4.3	Déploiement avec trois nœuds physiques	49
4.4	Conclusion	56
5	Déploiement cloud	57
5.4	Conclusion	57
5.1	Introduction	58
5.2	Avec amazon web services (AWS)	58
5.3	Avec microsoft azure	73
	Conclusion Générale	83

List of Figures

1.1	: Cycle de vie.	3
1.2	: Cycle de vie DevOps.	5
1.3	: Déploiement.	6
1.4	: Relation entre MLops et DevOps.	7
1.5	Recreate	9
1.6	Ramped	9
1.7	Bleu/Vert	10
1.8	Canary et Tests A/B	10
1.9	Tableau comparatif	11
2.1	Clustering	14
2.2	Différence entre une livraison et un déploiement	15
2.3	Architecture monolithique et microservice	16
2.4	Schéma de l'application à deployer	18
3.1	Cloud service models	20
3.2	machine virtuelle et conteneur	22
3.3	Architecture docker	23
3.4	Virtualisation	24
3.5	Architecture kubernetes	26
3.6	Worker node	27
3.7	Pods et nodes	28
3.8	Cluster IP	29
3.9	Node port	29
3.10	Load balancer	30
3.11	Ingress	30
3.12	Architecture kubeflow	32
4.1	DockerFile	35
4.2	Création de l'image	35
4.3	Déploiement avec Docker	36
4.4	Déploiement avec Kubernetes	36
4.5	Déploiement avec Kubernetes	37
4.6	Erreur	38
4.7	Deux images officiel	38
4.8	Docker Engine	39
4.9	Distribution linux sous windows	40
4.10	CRI containerd	40
4.11	Fichiers yaml des deux images et lancement du déploiement	41
4.12	Resultat	42
4.13	les repos git	43

4.14	Image de l'application	43
4.15	Image de l'application	44
4.16	Coturn	45
4.17	Kubeflow prérequis.	46
4.18	vps et microk8s	46
4.19	Installation de microk8s	47
4.20	Déploiement de kubeflow	48
4.21	Interface kubeflow	49
4.22	Installation et configuration de Kubeadm	49
4.23	Installation et configuration de Kubeadm	50
4.24	Kubeadm	50
4.25	Kubeadm	51
4.26	Helm	52
4.27	Installer Helm	53
4.28	Déployer le dashboard kubernetes	54
4.29	Déployer le dashboard kubernetes	55
4.30	Dashboard kubernetes	56
5.1	AWS Kubernetes	58
5.2	Amazon Eks	59
5.3	Crée un vpc	60
5.4	Crée un vpc	61
5.5	Créer un cluster	62
5.6	Créer un cluster	63
5.7	Amazon EC2	63
5.8	IAM	64
5.9	IAM	65
5.10	Instance EC2	66
5.11	Instance EC2	67
5.12	Les trois instances crée	68
5.13	Installation sur les trois instances	69
5.14	Installation sur les trois instances	70
5.15	Installation sur les trois instances	71
5.16	Installation sur les trois instances	72
5.17	Types d'instances	72
5.18	Crée un cluster avec l'interface Azure	73
5.19	Crée un cluster avec Azure CLI	74
5.20	Le cluster crée	75
5.21	Se connecter au cluster	75
5.22	Deux méthodes de déploiement	76
5.23	Méthode une	76
5.24	Méthode une étapes	77
5.25	Méthode deux	78
5.26	Méthode deux étapes	79
5.27	Déployer le front	80
5.28	Déployer le coturn	81
5.29	Etapes du déploiement de kubeflow sur azure	81
5.30	Déploiement de kubeflow sur azure	82

Liste des abréviations

Dev	développeurs
Ops	opérationnels
KL	Machine learning
GPU	Graphics Processing Unit
CPU	Central Processing Unit
K8s	Kubernetes
CLI	Command Line Interface
VDI	Virtual desktop infrastructure
API	Application Programming Interface
VM	Virtual Machine
CD	Continuous delivery
CI	Continuous integration
IP	Internet Protocol
OS	Operating system
Iaas	Infrastructure as a service
Paas	Platform as a service
Saas	Software as a service

Introduction Générale

L'intelligence artificielle (IA) est un processus qui imite l'intelligence humaine, basé sur la création et l'application d'algorithmes qui s'exécutent dans un environnement informatique dynamique, son but est de permettre aux ordinateurs de penser et d'agir comme des êtres humains. Cela dit, comme on pouvait s'y attendre, la pensée humaine est non seulement complexe mais aussi assez versatile ce qui rend la tâche de sa traduire en algorithmes extrêmement fastidieuse voir quasiment impossible.

C'est ici que le Machine Learning (ML) entre en jeu, ayant pour but et objectif de donner aux ordinateurs la possibilité et la capacité d'apprendre par eux-mêmes simplement en se basant sur des données numériques massives (Big Data) qu'ils traitent. Cela permet la création d'applications d'apprentissages automatiques pouvant analyser et assimiler un volume énorme de données en un temps record prenant avantage du fait que la vitesse de traitement d'un humain restera toujours inférieure à celle d'un ordinateur moderne et ce fossé ne cesse de se creuser aux avancées dans le domaine de l'électronique.

Le Machine Learning a introduit un nouveau chapitre dans le domaine de l'automatisation, en d'autres termes il a permis l'automatisation et l'optimisation de plusieurs tâches ce qui a entraîné le développement fulgurant de certains domaines tels que l'automobile (voitures intelligentes), l'industrie lourde (machines automatiques), la santé ... etc.

Cependant les scientifiques des données créent et développent une variété de modèles différents, dont 87% n'arrivent pas au stade de déploiement[1]. La mise en production et la maintenance des modèles d'apprentissage automatique sont l'un des défis les plus pressants auxquels les organisations sont confrontées aujourd'hui. Le flux de travail ML, qui comprend la formation, la construction et le déploiement de modèles d'apprentissage automatique, peut être un long processus avec de nombreux obstacles en cours de route[2].

Les conteneurs contiennent tous les éléments nécessaires au fonctionnement du code d'apprentissage automatique, ce qui garantit un environnement cohérent. La solution passe donc par un paradigme d'orchestration de conteneurs tels que Kubernetes qui contribue à l'automatisation de la gestion des conteneurs, comme la surveillance, la programmation et la mise à l'échelle.

Cet ouvrage couvre de prime abord les trois premiers chapitres qui décrivent tout ce qui touche au développement, au déploiement, au machine learning, à la conteneurisation et aux outils à utiliser. Le 4ème est dédié au déploiement local avec minikube et kubeadm, enfin il se termine par le déploiement cloud avec AWS et Azure.

Chapter 1

Développement et déploiement

1.1 Introduction

Nous présentons dans ce premier chapitre les concepts liés Aux phases du déploiement. Nous allons ainsi aborder des concepts tels que : Virtualisation, conteneurs, Kubernetes, etc. L'objectif principal est de décrire l'impact de ces techniques dans la phase de déploiement d'une application.

1.2 Développement logiciel

Pour réaliser une application, un logiciel ou un système informatique il faut impérativement passer par les phases de cycle de vie du développement logiciel. Afin de garantir une production réussie d'une solution de haute qualité qui répondra ou dépassera les exigences d'un client à toutes ses phases, dans le respect du budget et des délais prévus.

phases du développement logiciel :

- **Analyse des besoins** : Cette étape consiste à collecter tous les besoins du client (fonctionnels ou non fonctionnels), établir l'étude de l'existant (étude des produits similaires dans le marché), et analyse de faisabilité tout en déterminant les délais et les coups.
- **Planification** : Se résume à établir un concept ou une idée sur laquelle le logiciel sera développé. (planifier le calendrier de travail, les outils, les méthodes à utiliser et les risques auxquels ils pourraient être confrontés.)
- **Conception** : il s'agit de concevoir le système qui doit répondre aux exigences et de définir son architecture et les différents composants nécessaire.[3]
- **Développement** : Comprend l'étape où les programmeurs transforment des solutions proposées lors de la conception en un code opérationnel.
- **Tests** : Il s'agit de la phase de test et de validation qui déterminent les bugs techniques ou fonctionnels et la qualité du logiciel.

- **Déploiement et maintenance** : Le déploiement consiste à transférer le logiciel depuis l'environnement de développement vers l'environnement de production. Les utilisateurs pourront ainsi commencer à l'utiliser. La maintenance

qui s'effectue après la livraison du produit au client comprends la correction des erreurs et des anomalies du système, modification du système en ajoutant ou supprimant des fonctionnalités.

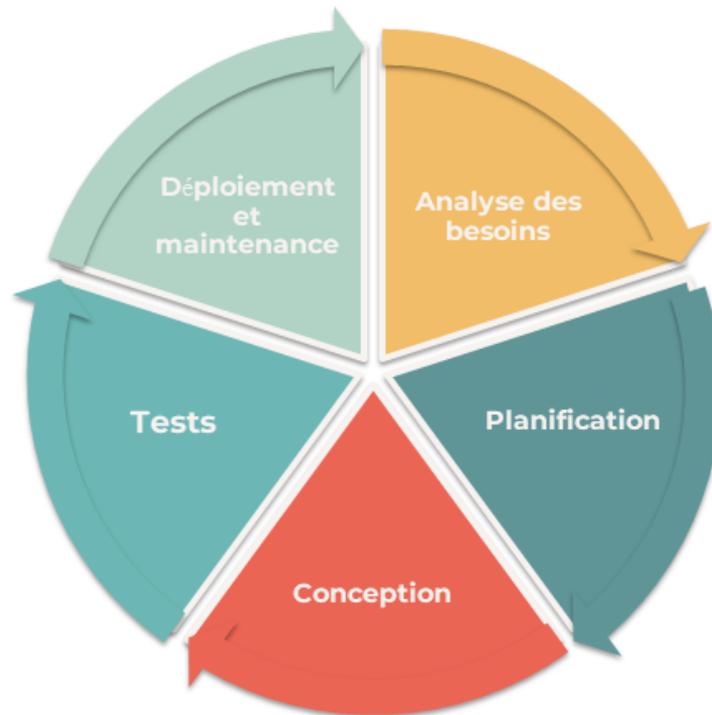


Figure 1.1: : Cycle de vie.

1.3 Concept du DevOps:

Commençant par comprendre comment les entreprises fonctionnaient traditionnellement sans DevOps.

Les développeur ou les devs recueillent les exigences du métier avant de développer le code, ensuite cette même équipe teste le nouveau code avant de le livrer à l'équipe qui se chargera de la production et d'exploitation Qu'on appelle Ops pour opérations. Cependant ce modèle pose des problèmes car les deux équipes ont des objectifs différents :

Les "dev" ont pour but d'obtenir un "time to market réduit", le temps entre la conception du produit et la mise en production. Un time to market rapide permet à une entreprise de s'adapter et de répondre plus rapidement à la demande du marché mais pour aussi proposé de l'innovation aux clients avant les concurrents. De l'autre côté l'Objectif de l'équipe Ops est de faire garantir la stabilité du système cela passe donc par un contrôle très sévère, Une panne de production pouvant couter très très cher à l'entreprise. Cet antagonisme provoque des conflits d'intérêts, les devs blâment les retards et les problèmes de livraison des Ops et les Ops blâment les incidents de l'équipe de production liés à la mauvaise qualité du code.

Afin d'améliorer le processus précédent, la méthodologie DevOps a émergé pour créer un environnement dans lesquels la conception, les tests et la publication de logiciels peuvent

se produire rapidement, fréquemment et efficacement en alignant les objectifs à court terme autour d'un objectif commun sur le long terme.

- Le cycle de vie DevOps :

Le cycle de vie DevOps comprend six phases [4]

1. **Développement continue** : C'est la phase dans laquelle le logiciel est "planifié" et "codé". Il n'y a pas besoin d'outils DevOps pour la planification, mais il existe de nombreux outils de maintenance du code. Il est aussi possible d'écrire du code dans n'importe quel langage, mais il est géré par un outil de contrôle de version. La maintenance du code est connue sous le nom de contrôle du code source. Les outils les plus couramment utilisés sont Git, SVN, Mercurial, CVS et JIRA.
2. **Intégration continue (CI)** : Cette phase est au cœur de l'ensemble du cycle de vie DevOps. Il s'agit d'une pratique de développement logiciel dans laquelle les développeurs doivent apporter des modifications au code source plus fréquemment. Cela peut être fait quotidiennement ou hebdomadairement. Cela permet d'identifier rapidement les problèmes potentiels. L'écriture de code comprend non seulement la compilation, mais également la révision du code, les tests unitaires, les tests d'intégration et l'emballage. Le code prenant en charge les nouvelles fonctionnalités continuera d'être intégré au code existant. Comme le développement logiciel est continu, le code mis à jour doit être intégré en continu et en douceur dans le système pour refléter les changements de l'utilisateur final.
3. **Tests continus** : C'est l'étape au cours de laquelle le logiciel développé est constamment testé pour détecter les erreurs. Des outils de tests automatisés sont utilisés qui permettent de tester plusieurs bases de code de manière approfondie en parallèle pour s'assurer qu'il n'y a pas de défauts dans les fonctionnalités. À ce stade, il est possible d'utiliser le conteneur Docker pour simuler l'environnement de test. L'automatisation des tests fait gagner beaucoup de temps, d'efforts et de travail pour exécuter les tests au lieu de les exécuter manuellement. Le rapport est également un gros plus. Il est également possible de planifier l'exécution du scénario de test à une heure prédéfinie. Après les tests, le code continuera à être intégré dans le code existant.
4. **Déploiement continu (CD)**: C'est à ce moment que le code est déployé sur le serveur de production. Il faut s'assurer que le code est correctement déployé sur tous les serveurs. Dans cette phase il y a un ensemble d'outils qui permet un déploiement continu.
 - * Outils de gestion de la configuration est le processus d'établissement et de maintien de la cohérence entre les exigences fonctionnelles et les performances de l'application. C'est-à-dire le déploiement, la planification des mises à jour pour tous les serveurs et plus important encore, le maintien de la cohérence de la configuration sur tous les serveurs.
 - * Les outils de conteneurisation aident à maintenir la cohérence entre les environnements dans lesquels l'application est développée, testée et déployée. Grâce à ces outils, l'intégration et reproduction des mêmes dépendances et packages

utilisés dans le développement, les tests et la préparation, il n'y a donc aucun risque d'erreur ou d'échec en production.

5. **Surveillance et alerte:** Il s'agit d'une étape très importante du cycle de vie "DevOps" qui surveille en permanence les performances des applications. Des informations importantes sur l'utilisation du logiciel sont stockées ici. Ces informations sont traitées pour reconnaître le bon fonctionnement de l'application, les erreurs système telle que mémoire insuffisante, serveur inaccessible, etc. Dans cette phase, la cause de chaque problème est identifiée.

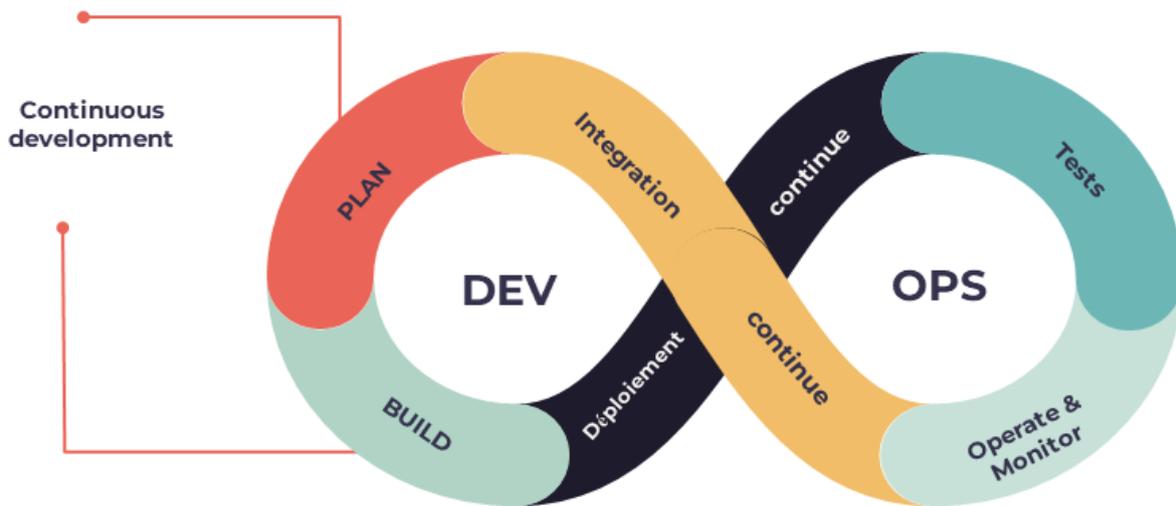


Figure 1.2: : Cycle de vie DevOps.
[5]

1.4 Déploiement d'applications

Le déploiement est le mécanisme par lequel les applications, les modules, les mises à jour et les correctifs sont livrés des développeurs aux utilisateurs.[6]

Expliquer de manière plus simple (voir figure 1.3):

Lorsqu'on crée une application, on y a accès que sur notre propre ordinateur ce qu'on appelle localhost (environnement de développement local), cette application est pas disponible au public. Si on veut la diffuser aux utilisateurs, On aura besoin de la rendre disponible sur des serveurs web qui hébergent notre code d'application, l'exécutent et le livrent a toutes personne qui le demande, Cela permet a d'autre ordinateurs sur internet d'y accéder. les différentes étapes nécessaires à la diffusion d'une application à un publique prédéfini est ce qu'on appelle le processus de déploiement.

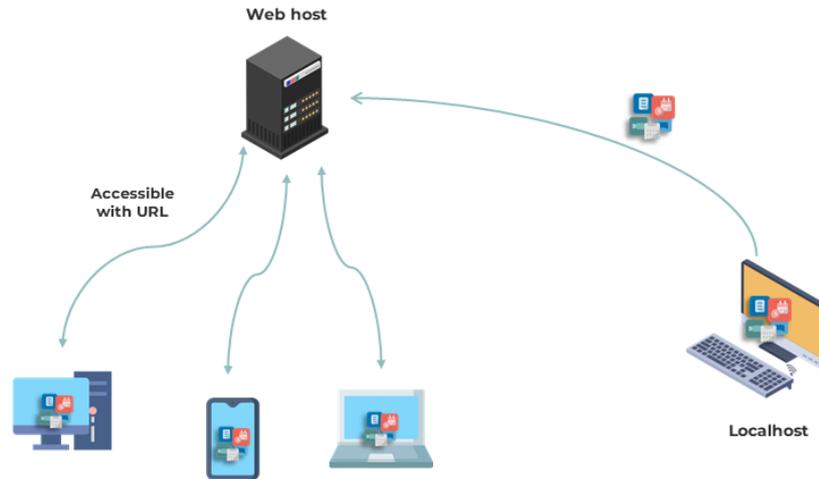


Figure 1.3: : Déploiement.

1.5 Type de déploiement

- **On premise:** Concerne le déploiement local où l'entreprise est propriétaire et responsable de la prise en charge de l'infrastructure et des serveurs dédiés au système, Son installation comprend un investissement initial considérable, mais offre un contrôle complet sur la sécurité, la maintenance et les mises à jour.
- **On ligne:** Concerne le déploiement Cloud computing qui implique que le système soit hébergé par un fournisseur tiers, la solution est louée sur une base horaire, journalière, mensuelle ou annuelle. Le fournisseur est responsable de la sécurité, Les mises à jour et la planification des patches.

- Les modèles de déploiement du Cloud Computing comprennent:

- (a) **Cloud public:** Les ressources sont ouvertes au grand public, et sont partagées avec d'autres entreprises, ce qui diminue les coûts et la sécurité
- (b) **Cloud privé:** Est plus cher car l'ensemble des ressources sont réservées à l'usage exclusif d'une seule entreprise, ne sont donc pas partagées avec d'autres firmes, assurant ainsi un plus grand contrôle sur la vitesse et la sécurité.
- (c) **Cloud hybride:** Correspond à un mélange entre le cloud privé et public c'est-à-dire avoir des ressources dédiées en interne et des ressources publiques en externe

Dans ce qui suit nous allons parler d'un concept associé au DevOps et qui est MLOps.

1.6 MLOPS

MLOps ou Machine Learning Operations est l'équivalent ML de DevOps qui est un ensemble d'approches de gestion du cycle de vie ML pour rationaliser, automatiser et accélérer le déploiement de modèles ML afin d'aider les équipes Data Science (DS), les ingénieurs DevOps et les informaticiens.[7]

1.6.1 Pourquoi avons-nous besoin de MLOps ?

L'introduction de l'apprentissage automatique dans un environnement de production est difficile. Le cycle de vie de l'apprentissage automatique comprend de nombreux composants complexes tels que: l'ingestion de données, la préparation des données, la formation de modèles, l'optimisation de modèles, le déploiement de modèles, la surveillance de modèles et la responsabilité.

Nous avons également besoin de collaboration et de transfert entre les équipes, de l'ingénierie des données à la science des données en passant par l'ingénierie ML.[8]

-Les phases clés de MLOps sont les suivantes :

1. **Création:** Comprend la préparation des données, l'ingénierie des fonctionnalités, la création de modèles et les tests.
2. **Gestion:** une fois les modèles créés, ils sont généralement placés dans un référentiel vérifiable soumis au contrôle des versions afin de favoriser leur réutilisation dans l'ensemble de l'entreprise.
3. **Déploiement:** exportation, déploiement et intégration du modèle ou du pipeline à des systèmes et applications de production.
4. : une surveillance continue est nécessaire pour garantir des performances optimales. À mesure que les données changent, le modèle peut être réentraîné ou remplacé par un nouveau modèle.

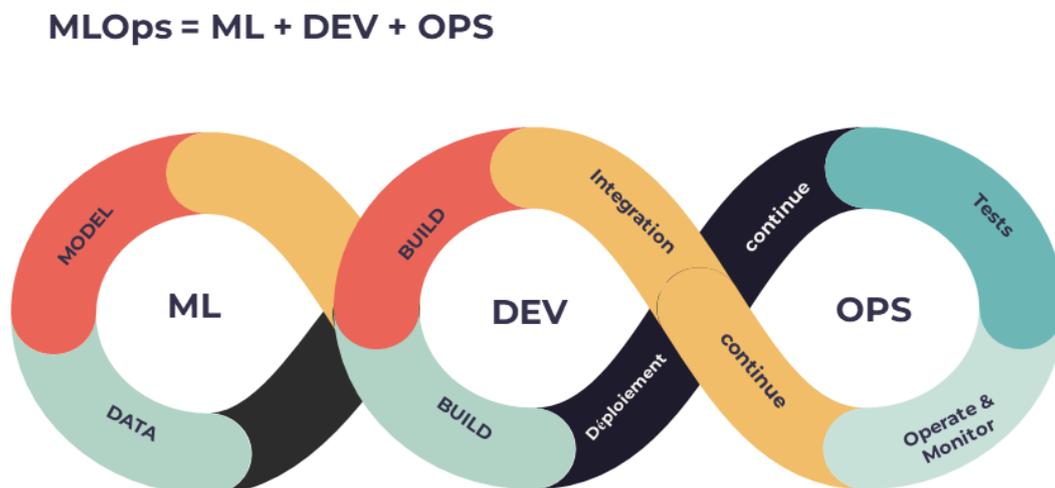


Figure 1.4: : Relation entre MLOps et DevOps.

1.7 Stratégie de déploiement

Une stratégie de déploiement est un moyen de modifier ou de mettre à niveau une application. L'objectif est d'apporter des modifications sans temps d'arrêt de manière à ce que l'utilisateur remarque à peine les améliorations. Il est donc important de bien choisir la bonne stratégie.[9]

1. **Recreate:** Consiste à désactiver et arrêter la version A puis à déployer la version B (voir figure 1.5)
2. **Ramped(rolling):** La version B est lentement déployée et remplace la version A, Les déploiements par phases impliquent la mise à jour d'un sous-ensemble de versions d'application (plutôt que la mise à jour d'un déploiement de base complet), Un ReplicaSet secondaire est créé avec la nouvelle version de l'application, puis le nombre de répliques de l'ancienne version est réduit et la nouvelle version est augmentée jusqu'à ce que le nombre correct de répliques soit atteint (voir figure 1.6).
3. **Bleu/Vert :** Est une stratégie pour déployer deux versions d'une application en même temps. Un environnement (bleu) exécute la version actuelle de l'application et un environnement (vert) exécute la nouvelle version de l'application. La version bleue continue de fonctionner tandis que la version verte subit des tests (voir figure 1.7).
4. **Canary:** La version B est publiée pour un sous-ensemble d'utilisateurs, puis procède à un déploiement complet permettant de réduire le risque d'introduction d'une mise à jour logicielle en production (voir figure 1.8).
5. **Tests A/B:** La version B est diffusée à un sous-ensemble d'utilisateurs dans des conditions spécifiques. En bref, testez une nouvelle fonctionnalité avec un ensemble ciblé (et non aléatoire) d'utilisateurs, mais en fonction de segments de paramètres utilisateur courants tels que la géographie ou le type d'appareil (voir figure 1.9).

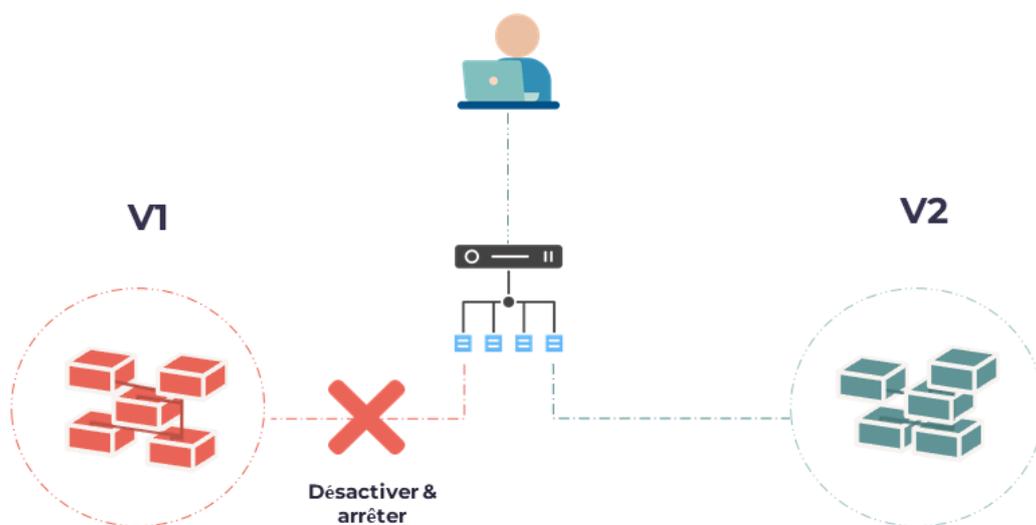


Figure 1.5: Recreate
[9]

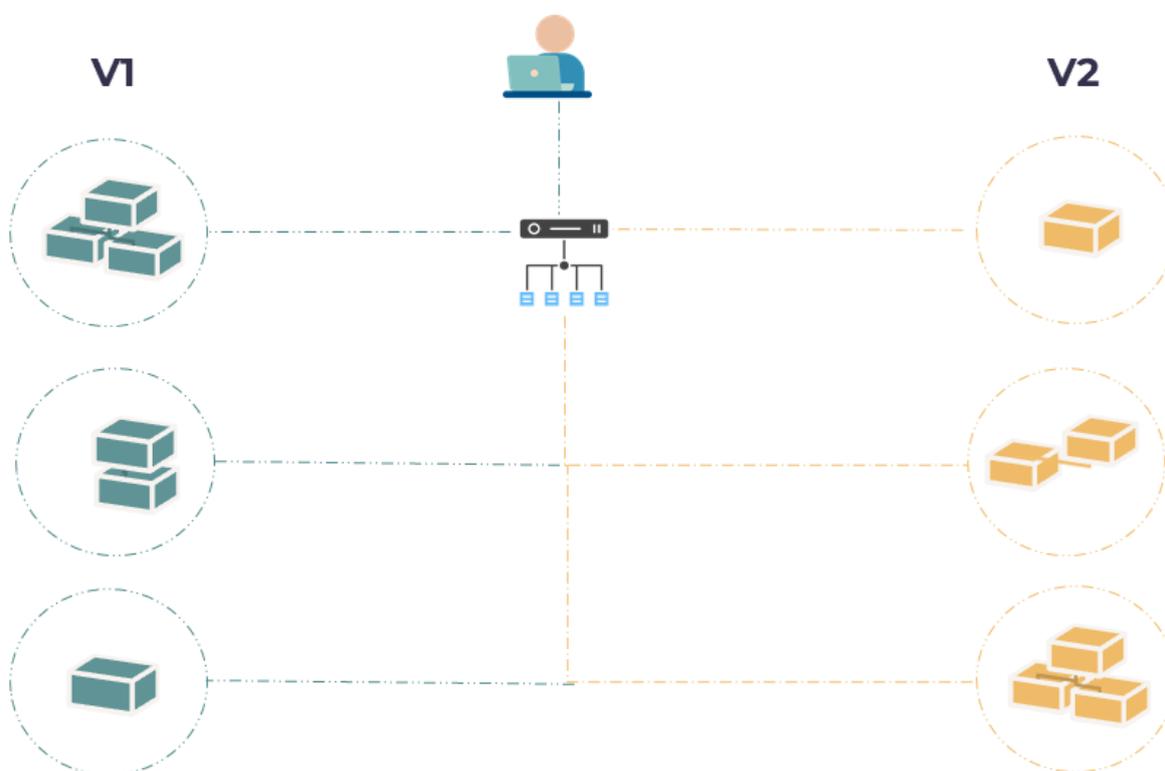


Figure 1.6: Ramped
[9]

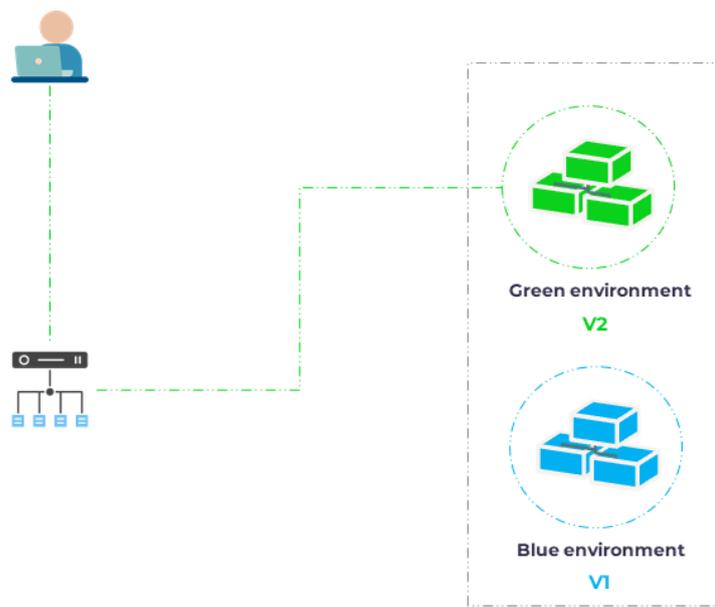


Figure 1.7: Bleu/Vert [9]

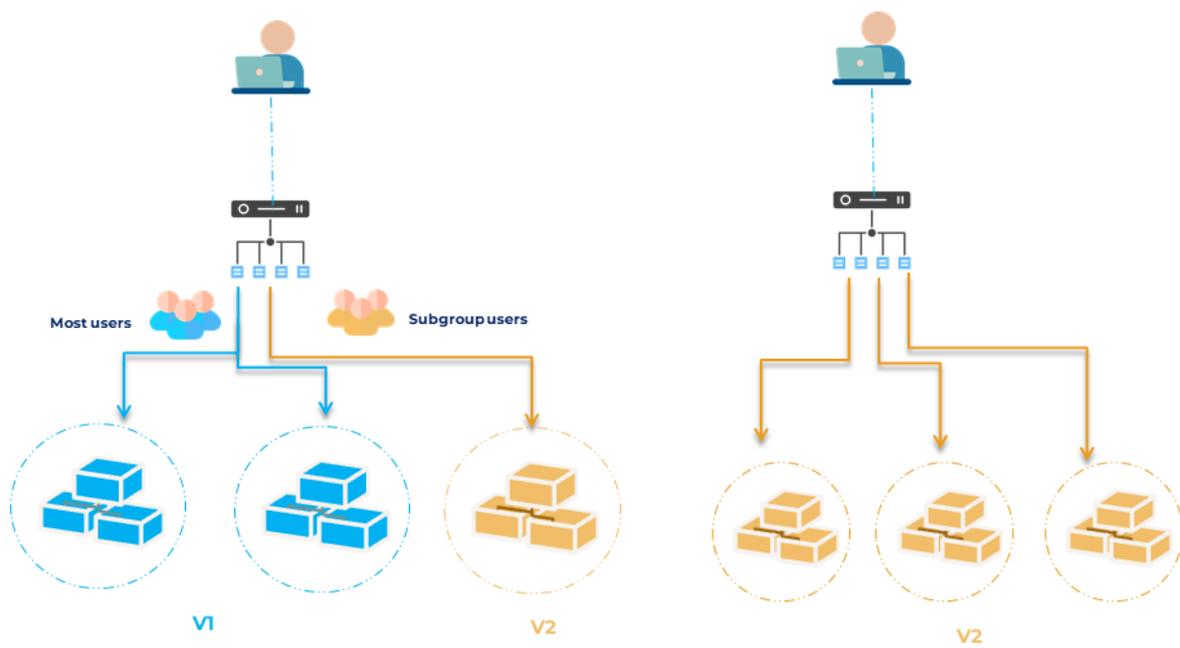


Figure 1.8: Canary et Tests A/B [9]

STRATEGY	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE	✗	✗	✗	★	★★★★	★★★★	-
RAMPED	✓	✗	✗	★	★★★★	★	★
BLUE/GREEN	✓	✗	✗	★★★	-	★★	★★
CANARY	✓	✓	✗	★	★	★	★★
TESTS A/B	✓	✓	✓	★	★	★	★★★

Figure 1.9: Tableau comparatif
[9]

1.8 Conclusion

Dans ce premier chapitre nous avons expliqué quelques généralités en relation avec notre thème, Nous avons pu voir ce que sont le devops, le mlops et le déploiement.

Dans le prochain chapitre nous allons exposer la problématique, les contraintes et les différentes propositions pour y remédier

Chapter 2

Machine Learning

2.1 Introduction

Dans ce chapitre nous allons découvrir la problématique que nous devrions résoudre au cours de ce travail, durant lequel nous serons amenés à analyser et à traiter le déploiement des modèles de machine learning.

2.2 Machine Learning

Lorsque nous interagissons avec les banques, achetons en ligne ou utilisons les réseaux sociaux, les algorithmes de machine Learning entrent en jeu pour optimiser, fluidifier et sécuriser notre expérience. Quand nous visitons un site Web complexe comme Facebook, Amazon ou Netflix, il est probable que chaque partie du site Web contienne un modèle d'apprentissage automatique (ML) qui permet d'analyser et tracker l'activité des clients afin d'améliorer l'expérience utilisateurs.

Le machine Learning une discipline scientifique centrée sur le développement, l'analyse et l'implémentation de méthodes automatisables, qui offrent la possibilité à une machine d'évoluer grâce à un processus d'apprentissage. En raison de son omniprésence croissante chaque compagnie est susceptible de le rencontrer et aura besoin de connaissances pratiques dans ce domaine. Un sondage Deloitte 2020 a révélé que 67% des entreprises utilisent l'apprentissage automatique et que 97% l'utilisent ou prévoient de l'utiliser au cours des prochaines années.[10]

Cependant, l'utilisation de l'apprentissage automatique nécessite beaucoup de ressources, une capacité de stockage énorme, une puissance de calcul élevée et une bande passante importante,ect.[11]

Dans ce qui suit nous allons citer certains mots clés relatifs au domaine ML:

1. **GPU:** Unité de traitement spécialisée avec une capacité de calcul mathématique améliorée, ce qui le rend idéal pour l'apprentissage automatique. les GPU dotés d'une mémoire plus grande et plus rapide, tels que le GPU NVIDIA A100 Tensor Core , sont capables de traiter plus rapidement des lots de données. L'utilisation de GPU pour la science des données réduit le temps de traitement de l'apprentissage automatique et augmente la vitesse de calcul, l'entraînement des modèles de Machine Learning peut être effectué 215 fois plus rapidement.

[12]

2. **CPU:** Unité central de traitement considéré comme le cerveau de l'ordinateur où la plupart des calculs sont effectués. En termes de puissance de calcul, le CPU est l'élément le plus important d'un système informatique. Les calculs de préparation et de prétraitement des données nécessaires à l'apprentissage du ML sont généralement effectués sur le CPU, bien que des innovations récentes aient permis d'en réaliser de plus en plus sur les GPU.
3. **Mémoire et stockage:** En particulier pour les plus grands modèles actuels, l'entraînement ML ne fonctionne que lorsqu'il y a une quantité extrêmement importante de données d'entrée à entraîner. Ces données sont extraites du stockage par lots, puis traitées par le CPU dans la mémoire du système avant d'être transmises au GPU.
4. **Réseau:** Les systèmes d'IA étant souvent regroupés en clusters pour faire évoluer les performances, ils disposent de plusieurs interfaces Ethernet de 10 gbps ou plus. Certains comprennent également des interfaces InfiniBand ou des interfaces GPU dédiées (NVLink) pour les communications intra-clusters.
5. **Cluster:** L'utilisation des ressources citées précédemment nécessite de les regrouper en cluster, qui désigne un groupe de serveur vu de l'extérieur comme un seul et même serveur logique, le cluster répond aux demandes de traitements d'applications continues au quel un seul serveur peut difficilement répondre et garantis les continuités des services.

Il y a 2 types de clustering:

- (a) **Actif/Passif:** Le principe est de doubler un serveur avec un second serveur, les deux sont démarrés mais seul un serveur traite les requêtes, Il répond a besoin de disponibilités mais de montée en charge (voir figure 2.1).
- (b) **Actif/Actif:** Le principe est La redondance du serveur actif ce qui fait que la charge de travail est répartie entre eux , Si l'un d'eux tombe ce sont les autres serveurs actifs qui doivent reprendre le relais, il permet de gérer la montée en charge mais la mise en place et plus complexe (voir figure 2.1).

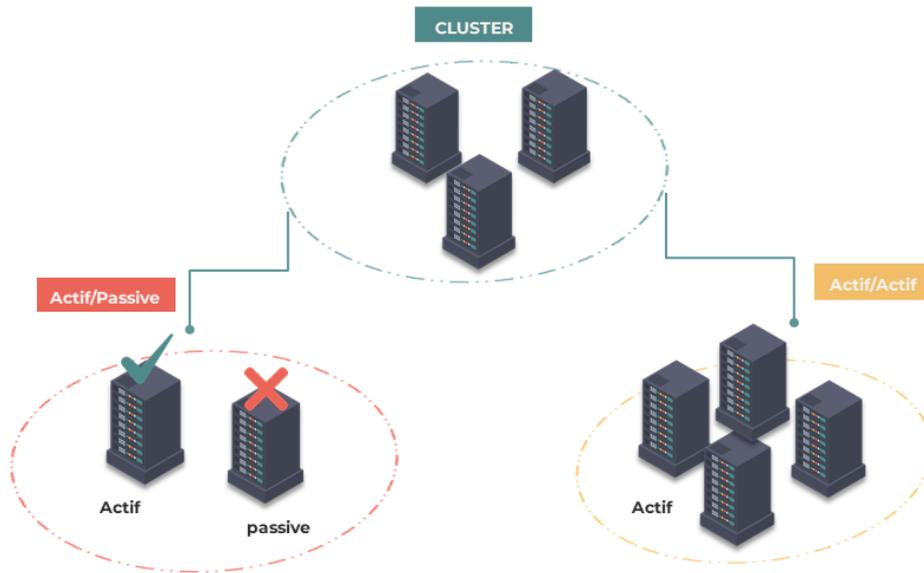


Figure 2.1: Clustering

Le déploiement des modèles de machine Learning est l'un des processus les plus difficiles pour tirer parti de l'apprentissage automatique. Cela nécessite une coordination entre les scientifiques des données, les équipes informatiques, les développeurs de logiciels et les professionnels de l'entreprise pour garantir que le modèle fonctionne de manière fiable dans l'environnement de production de l'organisation.[13]

2.3 Approche et contexte

Afin de garantir le succès lors des déploiements en production, chaque application nécessite un certain type et stratégie de déploiement cela dépend de l'architecture de son infrastructure qu'elle soit sur le cloud ou on premise, et des facteurs tels que le coût, les risques et le temps de déploiement.

2.4 Déploiement et releases

Le déploiement, comme nous l'avons expliqué précédemment, est défini comme la poussée de tous les composants d'une application d'un environnement à l'autre. Une release comprend le déploiement de l'ensemble de l'application ou de plusieurs applications intégrées dans un environnement de production.[14]

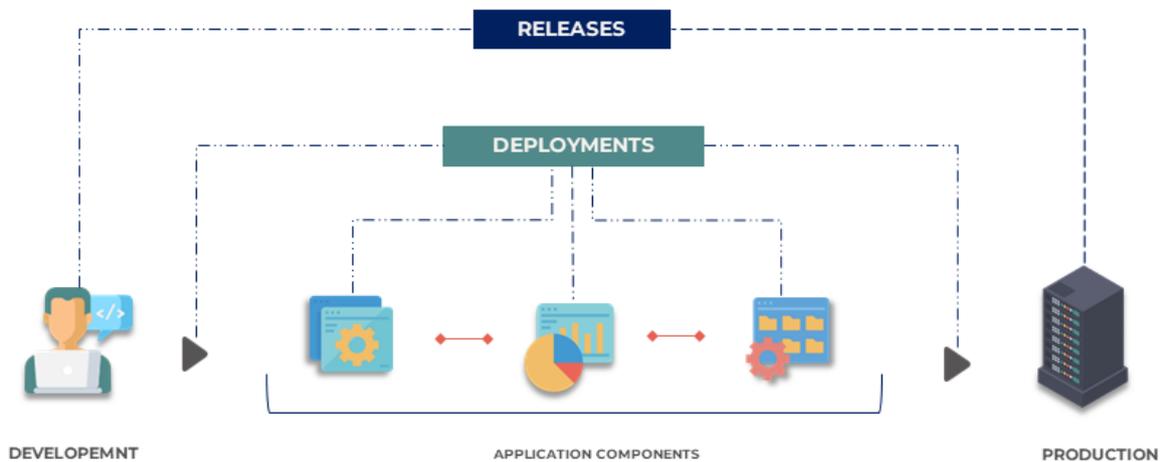


Figure 2.2: Différence entre une livraison et un déploiement

2.5 Services ou microservices

Les applications sont généralement créées de manière monolithique. Autrement dit, tous les éléments de l'application qui peuvent être déployés résident dans cette seule application. Un style architectural traditionnel et qui reste un bon choix pour de nombreuses applications. L'inconvénient, c'est que plus celle-ci est volumineuse, plus il devient difficile de l'enrichir de fonctions et de traiter rapidement les problèmes qui surviennent. Avec une approche basée sur des microservices, il est possible de résoudre ces problèmes, d'améliorer le développement et de gagner en réactivité.

Les microservices également connus sous le nom d'architecture de microservice sont un style architectural qui structure une application comme une collection de services qui sont : [15]

- Hautement maintenables et testables
- Faiblement couplés
- Déployables de manière indépendante
- Organisés autour des capacités de l'entreprise
- Détenus par une petite équipe

L'architecture de microservices permet la livraison rapide, fréquente et fiable d'applications volumineuses et complexes. Il permet également à une organisation de faire évoluer sa pile technologique.

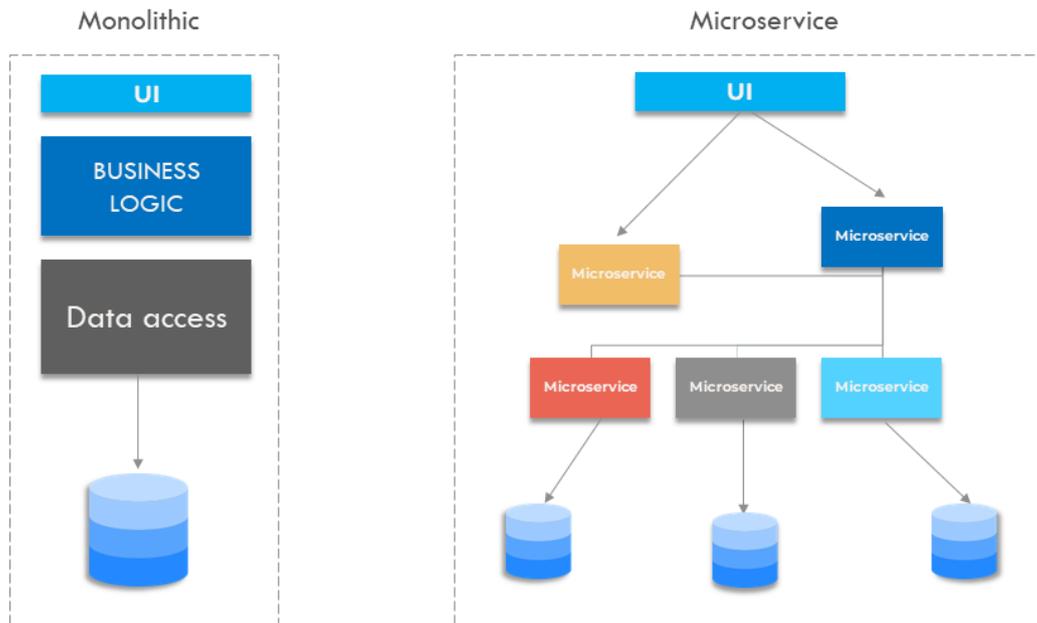


Figure 2.3: Architecture monolithique et microservice

2.6 Études des cas

Dans ce qui suit, nous présentons une étude de cas qui servira de scénario de déploiement pour le reste de ce document.

2.6.1 Scénario:

Dans le cadre des contrôles d'identité pour détecter les personnes recherchées, il est nécessaire de disposer de méthodes plus efficaces au-delà de l'utilisation de documents d'identification traditionnels tels que les permis de conduire et les passeports. Toutefois La reconnaissance faciale est l'une des technologies biométriques les plus répandue et performantes pour identifier un individu.

2.6.2 Secteurs exploitant la Reconnaissance facial

La reconnaissance faciale dans le monde réel, parmi ces secteurs on peut citer: [16]

1. **La Santé:** Afin de permettre aux médecins et aux hôpitaux d'inscrire plus facilement des patients sans avoir à s'aligner ou à interagir avec des formulaires.
2. **Les banques:** Commencent à utiliser la reconnaissance faciale comme outil de connexion sécurisé pour les clients utilisant leurs applications et sites Web bancaires en ligne
3. **Réseaux sociaux:** Comme DeepFace de facebook qui permet de reconnaître automatiquement les visages des personnes sur les photos téléchargées sur les réseaux sociaux.

4. **Éducation:** La numérisation du visage est utilisée dans certaines écoles pour rendre le campus plus sûr contre les menaces potentielles.
5. **Aéroport:** L'identification des passagers avec de faux documents tels que la reconnaissance faciale aux États-Unis a empêché un homme de tenter d'utiliser un faux passeport grâce aux proportions précises de l'appareil photo comparant des photos de visages de personnes enregistrées à partir de documents officiels du gouvernement.

2.6.3 Besoins liée au déploiement

Dans notre cas nous allons procéder au déploiement de la Solution cloud computing intelligent pour la reconnaissance en temps réel réalisé par le binôme Aitouakli Hichem et Mekhazni Fouad.

Une solution à grande échelle car ses fonctions de base nécessitent une grande puissance de calcul et une grande quantité de données qui évoluent en fonction des besoins des utilisateurs.

Parmi les fonctionnalités et besoins identifier, nous pouvons citer.

1. Détection des visages, sexe et objets en temps réel / Capture par webcam
2. Télécharger un modèle d'apprentissage automatique personnalisé
3. Performance
4. Portabilité
5. Scalabilité
6. Flexibilité
7. Sécurité
8. Robustesse

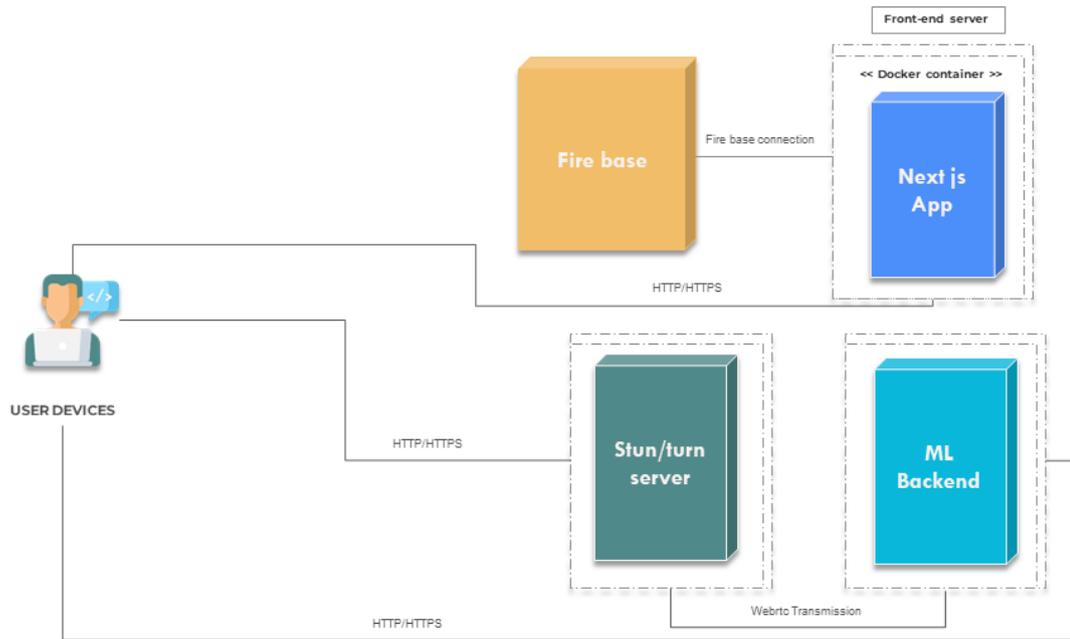


Figure 2.4: Schéma de l'application à déployer

2.6.4 solution proposée

La solution la plus attrayante que nous avons trouvée est le déploiement de ces applications via Cloud Computing.

2.7 Conclusion

Durant ce chapitre nous avons pu traiter la problématique autour de laquelle se construit notre travail, En expliquant les problèmes liés au déploiement complexe du machine Learning et à l'importance de sa mise en production. Quand au chapitre suivant, Celui-ci abordera certains éléments relatifs au cloud computing.

Chapter 3

Cloud computing

3.1 Introduction

Dans ce chapitre nous allons découvrir tous les outils que nous utiliserons au cours de notre travail, nous allons commencer par détailler le Cloud computing, puis nous suivrons avec la virtualisation et la conteneurisation, nous évoquerons aussi les outils relatifs à ces principes, docker et kubernetes, et enfin nous allons clore ce chapitre en décrivant la relation entre kubernetes et le ML ainsi que l'utilisation de kubeflow.

3.2 Cloud Computing

En termes simples, le cloud computing fournit des services informatiques via Internet (le "cloud") tels que des serveurs, du stockage, des bases de données, des réseaux, des logiciels, des analyses et des informations. Afin d'offrir une innovation plus rapide, des ressources flexibles et des économies d'échelle. En règle générale, on ne paye que pour les services en nuage que l'on utilise, ce qui aide à réduire les coûts d'exploitation et à gérer l'infrastructure plus efficacement.[17]

3.3 Modèles de services

Les modèles de services offerts par le cloud computing (voir figure 3.1) il y a: [17]

3.3.1 IaaS:

Infrastructure as a service, permet aux entreprises de louer des ressources informatiques (serveurs, réseaux, stockage, systèmes d'exploitation, etc...) Selon une tarification à l'usage.

3.3.2 PaaS:

Plateforme en tant que service, est un environnement de développement d'applications basé sur le cloud qui fournit aux développeurs tout ce dont ils ont besoin pour créer et déployer des applications. Cela inclut des outils de développement, des bibliothèques de code, des serveurs, des environnements de programmation et des composants d'application préconfigurés.

3.3.3 SaaS:

Software-as-a-service, le logiciel est hébergé sur un serveur distant et les clients peuvent y accéder à tout moment et en tout lieu, à partir d'un navigateur Web ou d'une intégration Web standard. Le fournisseur SaaS se charge des sauvegardes, de la maintenance et des mises à jour. Les solutions SaaS comprennent le progiciel de gestion intégré (ERP), la gestion de la relation client (CRM), la gestion de project, ect.

Cloud services models

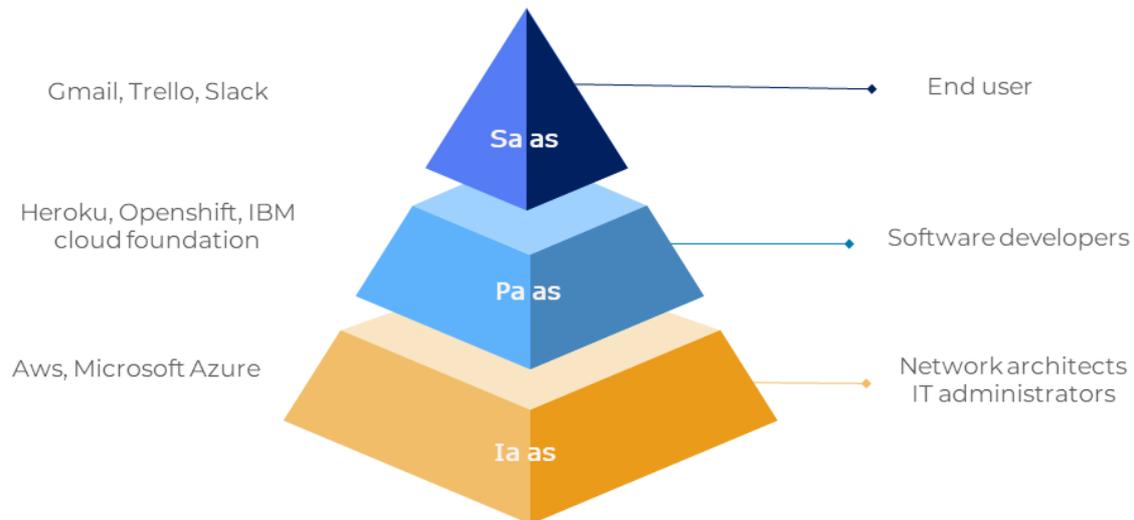


Figure 3.1: Cloud service models
[18]

3.4 Avantages du cloud computing

- **Coût:** Le Cloud Computing élimine les dépenses en capital liées à l'achat de matériel et de logiciels et à la configuration et à l'exploitation de centres de données sur site.
- **La rapidité:** Permet un déploiement rapide et une intégration facile. En effet, le déploiement de la solution cloud se fait en peu de temps et le service est disponible rapidement.
- **Sécurité et fiabilité:** De nombreux fournisseurs de cloud proposent un large éventail de politiques, de technologies et de contrôles qui renforcent globalement votre posture de sécurité, en aidant à protéger vos données, vos applications et votre infrastructure contre les menaces potentielles.
- **Flexibilité:** Les applications et services que vous utilisez dans le Cloud sont accessibles de n'importe où, tant que vous disposez d'un terminal et d'une connexion Internet. Le Cloud Computing s'adapte aux besoins évolutifs de votre entreprise en un temps-record.

- **Productivité:** Les plus grands services de cloud computing fonctionnent sur un réseau mondial de centres de données sécurisés, qui sont régulièrement mis à niveau vers la dernière génération de matériels informatiques rapide et efficace.

3.5 Virtualisation

Un mécanisme informatique qui se résume à faire fonctionner sur une même machine physique, plusieurs systèmes comme s'ils fonctionnaient sur des machines physiques distinctes. En d'autres termes la virtualisation est la création de plusieurs machines virtuelles (VM) à partir d'une seule machine physique à l'aide d'un logiciel appelé hyperviseur. La virtualisation permet donc une utilisation plus efficace du matériel informatique physique et offre un meilleur retour sur l'investissement matériel d'une organisation.

3.6 Machine virtuel et conteneur

3.6.1 Machine virtuel (VM)

Est une virtualisation au niveau matériel, Chaque VM exécute son propre système d'exploitation (OS) et se comporte comme un ordinateur indépendant, même si elle ne fonctionne que sur une partie du matériel informatique sous-jacent.

3.6.2 Conteneurisation

Un conteneur est similaire à une machine virtuelle mais la virtualisation s'effectue au niveau du système d'exploitation. Contrairement aux applications classiques, les applications conteneurisées partagent un environnement de système d'exploitation (noyau), elles utilisent donc moins de ressources que des machines virtuelles complètes, réduisent la pression sur la mémoire de l'hôte et sont compatibles avec les techniques émergents telles que le cloud computing, l'approche CI/CD et le DevOps.

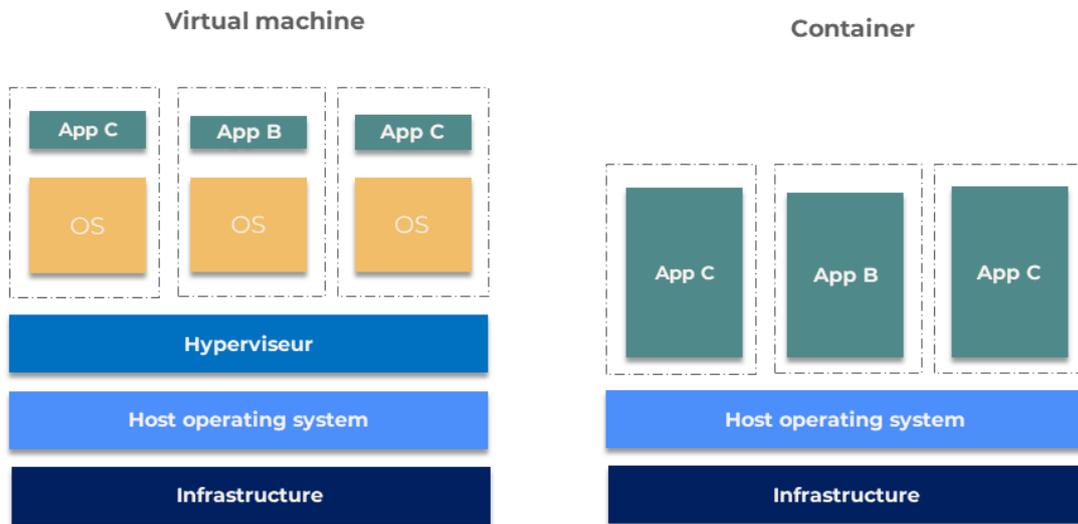
Les conteneurs sont si rapides et petits que nous pouvons les utiliser pour diviser une application monolithique en plusieurs petits services qui communiquent entre eux. Chaque petit service peut ainsi fonctionner dans son propre conteneur, séparément des autres, afin qu'il puisse être entretenu ou réparé sans craindre de briser les autres parties.

Parmi les systèmes de conteneurisations les plus populaire et utilisée et celui que nous allons aborder dans notre travail est le soi-disant Docker.

3.6.3 Docker

Docker est une plate-forme de conteneurs open source sécurisée et économique qui facilite la création de conteneurs et d'applications basées sur des conteneurs. On peut créer, tester, déployer et exécuter des applications sur n'importe quel environnement à l'aide de Docker. Cela évite les conflits potentiels au niveau de chaque composant et résout ainsi

les problèmes de version, d'incompatibilité ou encore d'implémentation qui surviennent souvent lors du développement de solutions.[19]



[20]

Figure 3.2: machine virtuelle et conteneur

3.6.4 Composants Docker

Dans ce qui suit nous allons décrire les différents composants docker. Dans ce qui suit nous allons décrire les différents composants docker [21]: (voir figure 3.3)

1. **Docker Engine:** Est le système qui fait tourner l'ensemble de Docker. Il s'agit d'un moteur qui suit une architecture client-serveur comprenant principalement trois composants :
 - **Serveur (docker daemon):** Il s'agit du démon docker appelé dockerd. Il peut créer et gérer des images docker, c'est-à-dire des conteneurs, des réseaux.
 - **Rest API:** Il est utilisé pour indiquer au démon docker ce qu'il doit faire.
 - **Interface de ligne de commande (CLI):** C'est un client qui est utilisé pour saisir les commandes du docker.
2. **Image docker:** Un modèle qui contient toutes les instructions pour créer le conteneur Docker. Elle est composée de plusieurs couches empaquetant toutes les installations, dépendances, bibliothèques, processus et codes d'application nécessaires pour un environnement de conteneur pleinement opérationnel.
3. **Registre docker:** Stocke les images Docker. Docker Hub est un registre public que tout le monde peut utiliser, et Docker est configuré pour rechercher des images sur Docker Hub par défaut. Vous pouvez même exécuter votre propre registre privé.

4. **Conteneur** : Les conteneurs sont des environnements encapsulés dans lesquels les applications sont exécutées. Un conteneur est défini par l'image et les options de configuration.

3.6.5 Architecture docker

En se référant à la documentation officielle. Docker utilise une architecture client-serveur. Le client Docker communique avec le démon Docker, qui effectue le gros du travail de création, d'exécution et de distribution de vos conteneurs. Le client et le démon Docker peuvent s'exécuter sur le même système, ou vous pouvez connecter un client Docker à un démon Docker distant.

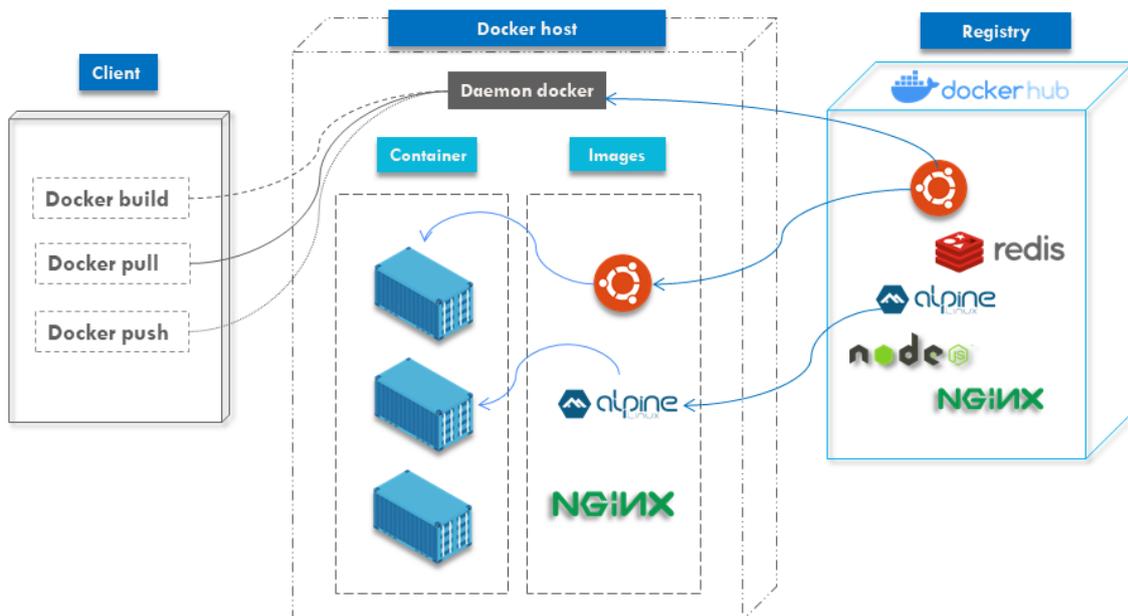


Figure 3.3: Architecture docker

3.7 Types de virtualisation

3.7.1 La virtualisation de systèmes d'exploitation:

Est une technique consistant à faire fonctionner en même temps, sur un seul ordinateur, plusieurs systèmes d'exploitation comme s'ils fonctionnaient sur des ordinateurs distincts.

1. **Virtualisation des serveurs:** Consiste à diviser un serveur physique en plusieurs serveurs virtuels uniques et isolés à l'aide d'un hyperviseur qui gère différents OS invités. Les multiples instances de serveur qui en résultent exécutent leurs propres systèmes d'exploitation indépendamment, s'exécutent en parallèle et ont des capacités différentes, mais exploitent le même matériel sous-jacent. Représentées de grands avantages aux entreprises : diminue le nombre de serveurs utilisés, optimises les ressources et réduit les coûts de matériel et maintenance.

2. **Virtualisation de poste de travail:** Comprend l'exécution de plusieurs systèmes d'exploitation de bureau, chacun sur sa propre machine virtuelle et sur le même ordinateur. L'un des principaux avantages de la virtualisation de bureau est que les utilisateurs peuvent accéder à tous leurs fichiers et applications personnels sur n'importe quel PC, ce qui signifie qu'ils peuvent travailler de n'importe où sans avoir besoin d'utiliser leur propre ordinateur de travail. Cela réduit également les coûts de licence et de mise à jour logicielle. La maintenance et la gestion sont simples car tous les postes de travail virtuels sont hébergés au même endroit.

Il existe deux types de virtualisation de postes :

- **VDI:** Virtual desktop infrastructure, La VDI consiste à abstraire et à héberger plusieurs sessions de bureau virtualisées sur un serveur principal centralisé. Ces bureaux virtuels sont ensuite mis à la disposition des utilisateurs accessibles via des terminaux clients légers.
- **La virtualisation de poste local:** Permet à l'utilisateur d'exécuter un ou plusieurs systèmes d'exploitation supplémentaires sur cet ordinateur et de passer d'un système d'exploitation à un autre selon les besoins, sans rien changer au système d'exploitation principal

3.7.2 La virtualisation d'application:

Permet d'exécuter des logiciels d'application sans installation directe sur le système d'exploitation de l'utilisateur. Cette méthode diffère de la virtualisation complète du poste de travail (mentionnée ci-dessus) car seule l'application s'exécute dans un environnement virtuel, le fonctionnement du système d'exploitation de l'appareil de l'utilisateur final étant inchangé.

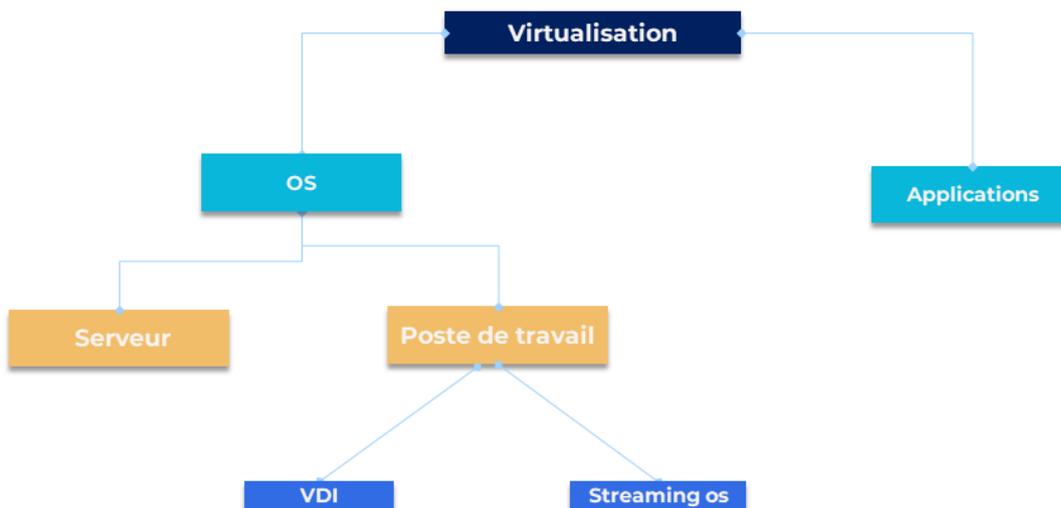


Figure 3.4: Virtualisation

3.8 Kubernetes (orchestration)

3.8.1 Définition

Également connu sous le nom de K8s, est un système open source permettant d'automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. conçu pour orchestrer des architectures de clusters de conteneurs logiciels de manière extrêmement agile, en donnant la possibilité de gérer l'allocation des ressources machine sous-jacentes à la volée.

Créé par Google, Développé en tirant les leçons de l'écriture et de l'exploitation du Project Borg pendant plus de 15 ans. un système de clustering utilisé par le géant américain depuis des années. Google Search, Maps, Gmail, Youtube, etc. Tous ses services reposent sur des clusters de centaines de containers pilotées par Borg, Par la suite Google en fait don à la CNCF (Cloud Computing Foundation). [22]

3.8.2 Fonctionnalités

Parmi les fonctionnalités de bases de K8s, nous citons:

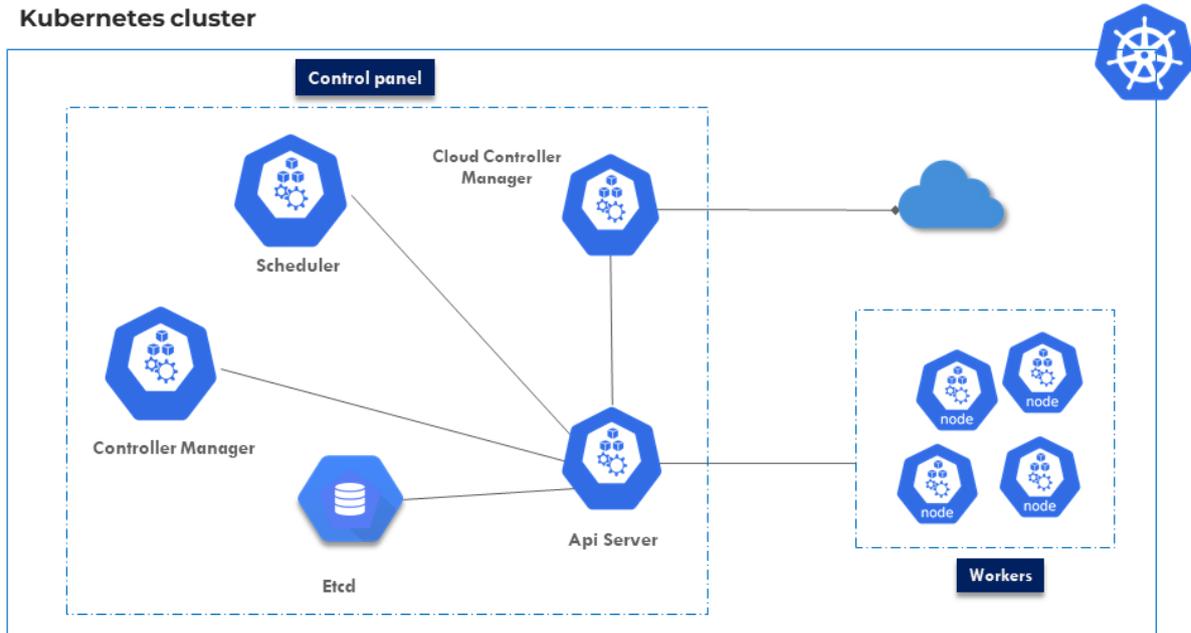
1. Orchestrer des conteneurs sur plusieurs hôtes
2. Optimiser l'utilisation de votre matériel afin de maximiser les ressources requises pour l'exécution de vos applications d'entreprise
3. Contrôler et automatiser les déploiements et mises à jour d'applications
4. Monter et ajouter des systèmes de stockage pour exécuter des applications avec état
5. Mettre à l'échelle des applications conteneurisées et leurs ressources à la volée.
6. Gérer des services de façon déclarative et garantir ainsi que les applications déployées s'exécutent toujours de la manière dont vous les avez déployées
7. Vérifier l'intégrité de vos applications et les réparer automatiquement grâce au placement, au démarrage, à la réplication et à la mise à l'échelle automatique

3.8.3 Architecture

Un environnement exécutant Kubernetes comprend les composants de base suivants (voir figure 3.5) 20:

- **Le nœud master (master node):** Exécute le plan de contrôle Kubernetes qui contrôle l'ensemble du cluster, composé d'un api-server, un Schedule, un etcd, un Controller managé et un Cloud Controller manager.
- **Le nœud de travail (worker node):** Se charge de l'exécution et de la mise en réseau des applications conteneurisées. Il effectue toutes les actions déclenchées via l'API Kubernetes, qui s'exécute sur le nœud maître. Se compose d'un servie kubelet, container runtime et d'un kube proxy.
- **Api-server:** Interface utilisée pour gérer, créer et configurer les clusters Kubernetes. C'est le moyen d'interaction entre les utilisateurs et le cluster Kubernetes.

Kubernetes cluster



[23]

Figure 3.5: Architecture kubernetes

- **Cluster store (etcd):** Base de données de Kubernetes où informations de configuration, d'exécution et d'état y sont stockées. C'est le seul composant "stateful" du master, il est crucial pour le bon fonctionnement de Kubernetes.
- **Scheduler:** Planificateur qui affecte les conteneurs à des nœuds en prenant en compte le taux de charge du nœud, sa capacité et les besoins en ressource.
- **Controller manager:** Exécute les contrôleurs qui surveillent en permanence l'état de votre cluster, puis effectuent ou demandent des modifications si nécessaire. Chaque contrôleur essaie de rapprocher l'état actuel du cluster de l'état souhaité.
- **Kubelet:** Permet la communication en worker et master node.
- **Container runtime:** Un environnement dans lequel les conteneurs peuvent être exécutés
- **Kubeproxy:** Chargé d'activer la communication entre les services au sein du cluster. [22]

Worker node

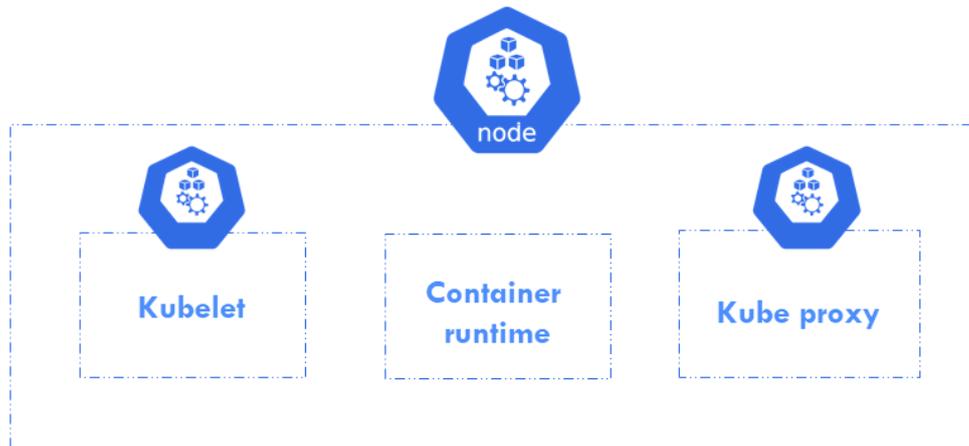


Figure 3.6: Worker node

3.9 Pods et services

Nous décrirons certains des concepts suivants concernant les k8 (voir figure 3.7)

3.9.1 Pods et nodes

1. **Pod:** Il s'agit de la plus petite unité de base d'une application Kubernetes, un pod peut contenir un unique conteneur (cas plus courant) ou plusieurs conteneurs (cas plus avancé), Au sein du système Kubernetes, les conteneurs d'un même pod partagent les mêmes ressources de calcul. Ces ressources de calcul sont regroupées pour former des clusters.
2. **Node:** Une machine de travail qui exécute des charges de travail Kubernetes. Il peut s'agir d'une machine physique ou d'une machine virtuelle (VM). Chaque nœud peut héberger un ou plusieurs pods.

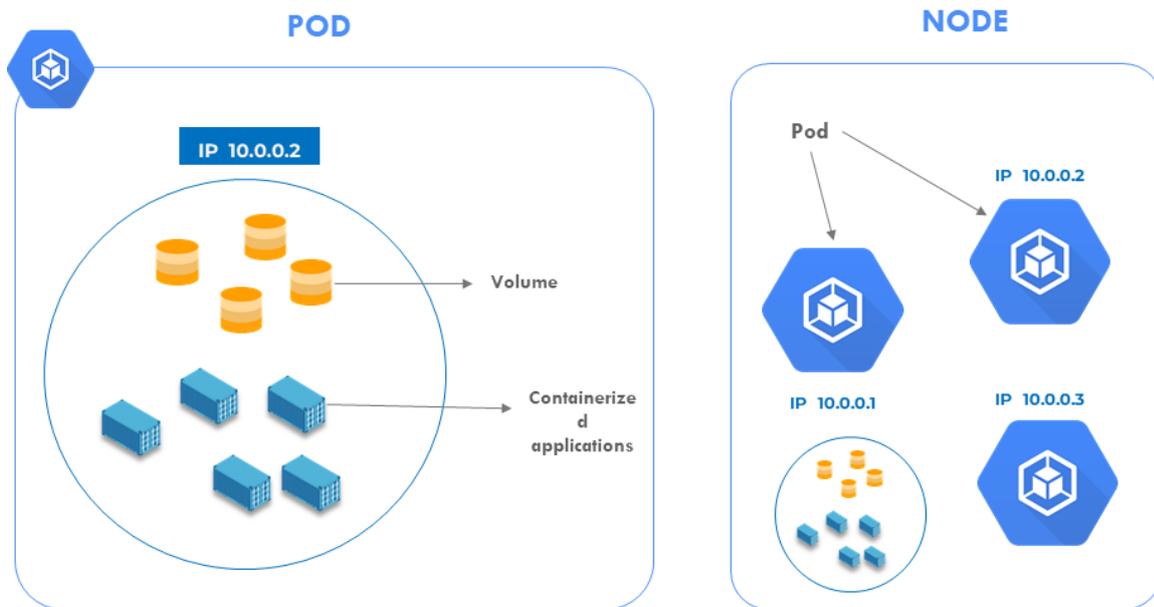


Figure 3.7: Pods et nodes

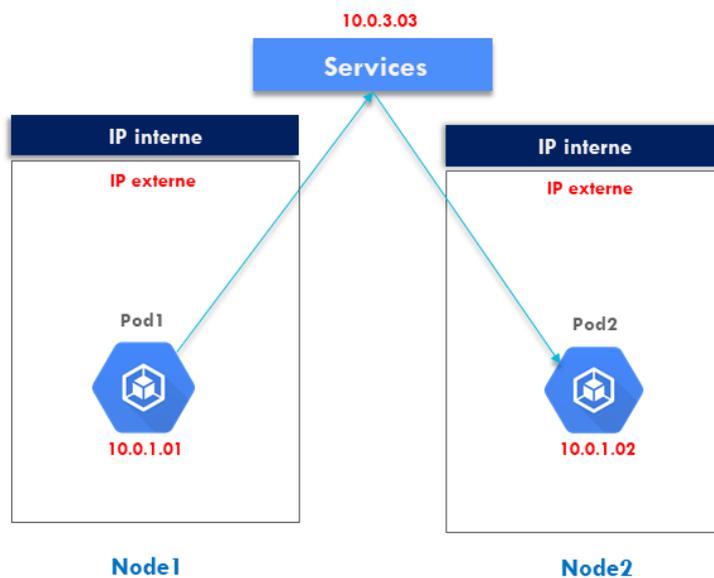
3.9.2 Services

Un service est chargé d'autoriser l'accès au réseau à un ensemble de pods. Il peut être défini comme une abstraction sur le dessus du pod qui fournit une adresse IP et un nom DNS uniques par lesquels les pods sont accessibles. Accorde aux applications de recevoir du trafic et peuvent être exposés de différentes manières et permet de gérer la configuration de l'équilibrage de charge.

Les Types possibles[24]:

- **Cluster IP:** Est le service Kubernetes par défaut. expose l'IP interne du cluster Kubernetes (voir figure 3.8).
- **NodePort:** Expose le Service sur l'IP de chaque nœud sur un port statique (voir figure 3.9).
- **LoadBalancer:** Expose le Service à l'extérieur en utilisant l'équilibreur de charge d'un fournisseur de cloud computing (voir figure 3.10)
- **Ingress:** Permet d'accéder à vos services depuis l'extérieur du cluster Kubernetes. IL est le plus utile si vous souhaitez exposer plusieurs services sous la même adresse IP (voir figure 3.11).

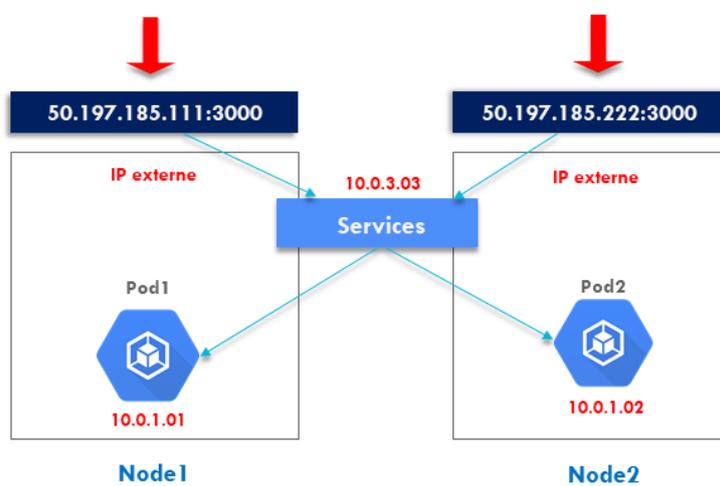
Cluster IP



[24]

Figure 3.8: Cluster IP

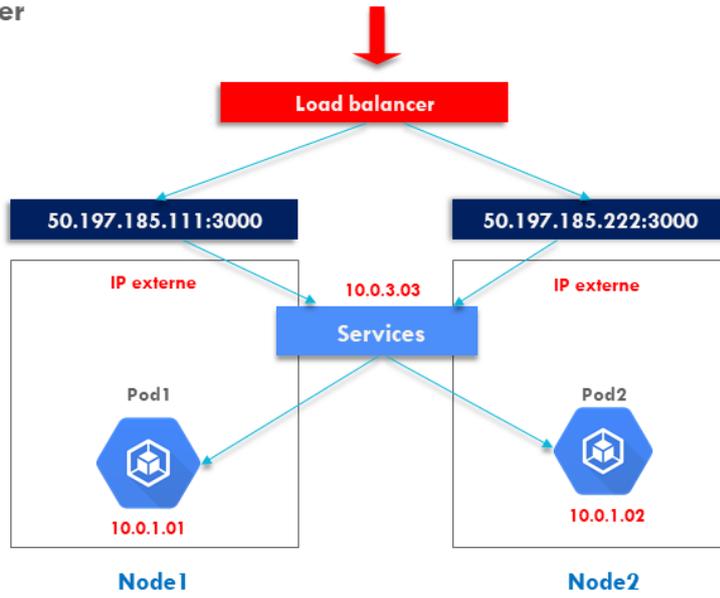
Node port



[24]

Figure 3.9: Node port

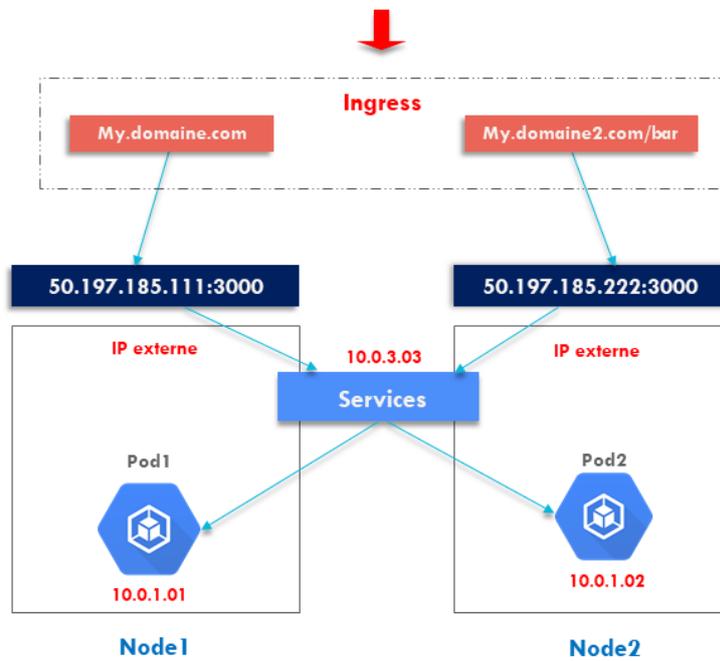
Load balancer



[24]

Figure 3.10: Load balancer

Ingress



[24]

Figure 3.11: Ingress

3.10 ML et kubernetes

Un des objectifs de notre projet est de déployer efficacement des modèles d'apprentissage automatique, les mettre à l'échelle, les distribuer sur des clusters de serveurs.

Kubernetes est l'un des moyens les plus populaires qui gère les modèles d'apprentissage automatique dans des conteneurs. Ces modèles peuvent être facilement mis à l'échelle et planifiés lorsqu'ils sont conteneurisés, la gestion des performances de la charge de travail peut être automatisée. À l'aide de Kubernetes, les organisations peuvent intégrer des flux de travail d'apprentissage automatique de bout en bout dans des conteneurs.

Une boîte à outils spécifique appelé Kubeflow a également été développées pour simplifier le processus de déploiement de modèles d'apprentissage automatique sur Kubernetes.

[25] [26]

3.10.1 Kubeflow

Sur la base des objectifs décrits précédemment, nous avons choisi Kubeflow comme plateforme pour nos services (voir figure 3.2).

Kubeflow est un projet open source dédié à rendre les déploiements de flux de travail d'apprentissage automatique (ML) sur Kubernetes simples, portables et évolutifs. Il ne vise pas à recréer d'autres services, mais à fournir un moyen simple de déployer les meilleurs systèmes open source pour le ML sur diverses infrastructures. Ainsi, partout où Kubernetes s'exécute, Kubeflow peut s'exécuter.

Kubeflow s'appuie sur la conteneurisation et le découplage en micro-services pour former une plateforme portable pouvant s'exporter sur n'importe quel environnement capable de déployer un cluster Kubernetes (notamment les principaux cloud providers).[27]

Kubeflow est livré avec de nombreux composants que l'on peut choisir d'installer ou non, et donc chacun a un rôle spécifique : [28]

- **Kubeflow pipeline:** Permet de construire, exécuter et administrer des pipelines ML automatisés.
- **Notebook server:** Permet de créer des serveurs Jupyter à la demande pour chaque utilisateur.
- **KFServing:** Utilisé pour exposer des modèles de Machine Learning.
- **Katib:** Utilisé pour optimiser les hyper-paramètres des modèles de Machine Learning en profitant de la puissance de calcul et du caractère distribué de Kubernetes.

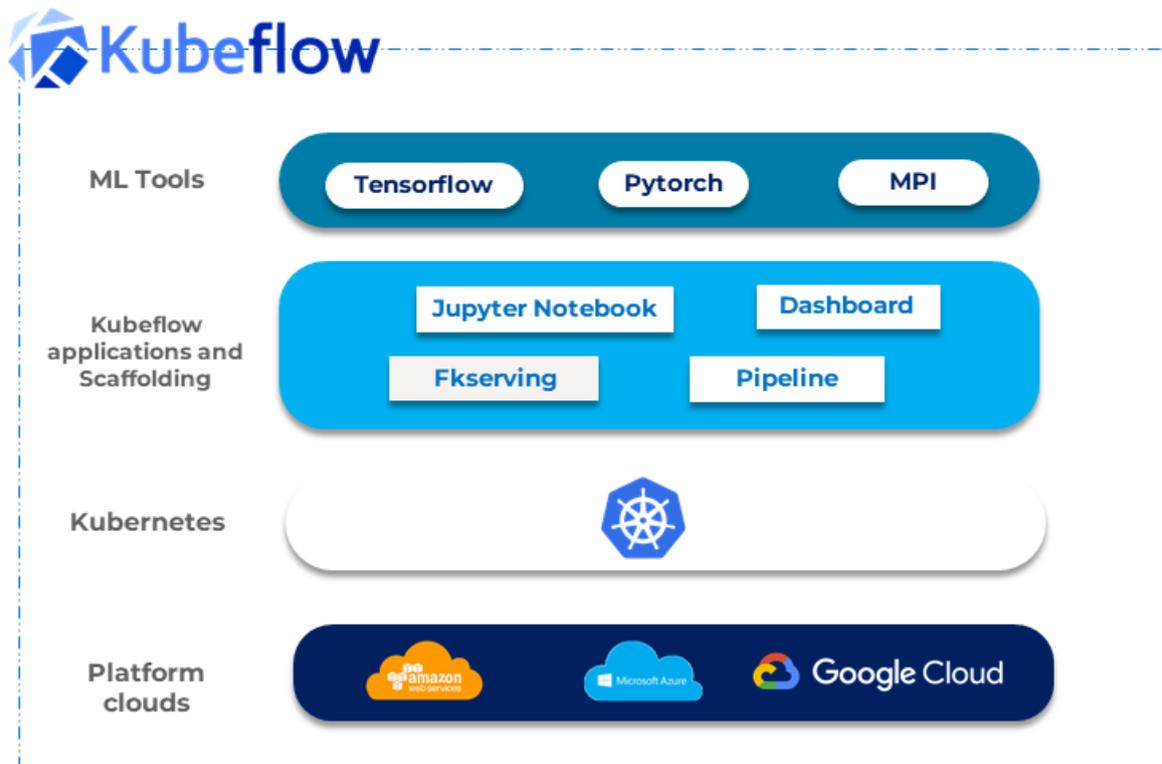


Figure 3.12: Architecture kubeflow

3.11 Conclusion

Jusqu'à présent, nous avons vu la plupart des concepts nécessaires liés aux outils que nous utiliserons dans ce travail, puis nous avons détaillé ces outils comme docker, kubernetes et kubeflow, et expliqué leurs composants. Dans le chapitre suivant, nous détaillerons la procédure de déploiement local via kubernetes.

Chapter 4

Déploiement local

4.1 Introduction

Dans ce chapitre, Nous allons démontrer une implémentation locale avec un seul nœud utilisant minikube et avec plusieurs nœuds utilisant kubeadm tout en précisant les problèmes rencontrés et les solutions adaptées.

4.2 Déploiement avec un seul nœud (Minikube)

Pour démarrer avec kubernetes, Minikube est l'outil idéal qui permet d'exécuter un cluster Kubernetes à nœud unique dans une machine virtuelle (VM) sur un ordinateur portable. Nous allons commencer par le déploiement d'une simple application appelé (getting-started) Qui s'agit d'une simple liste de tâches.

1. Installation:

Nous allons procéder a l'installation de MINikube, docker et Kubectl qui est l'outil de ligne de commande Kubernetes qui permet d'exécuter des commandes sur des clusters Kubernetes.

```
$ choco install kubernetes-cli
```

```
$ choco install minikube
```

```
$ choco install Docker-cli
```

```
$ choco install Docker
```

2. Création de l'image de notre application:

Le processus de création d'image kubernetes consiste a créé une image docker puis la pousser dans un registre ensuite la référencier depuis un pod.

Image kube



- (a) **Crée un dockerfile:** Simple document texte qui contient une série de commandes que Docker utilise pour créer une image.

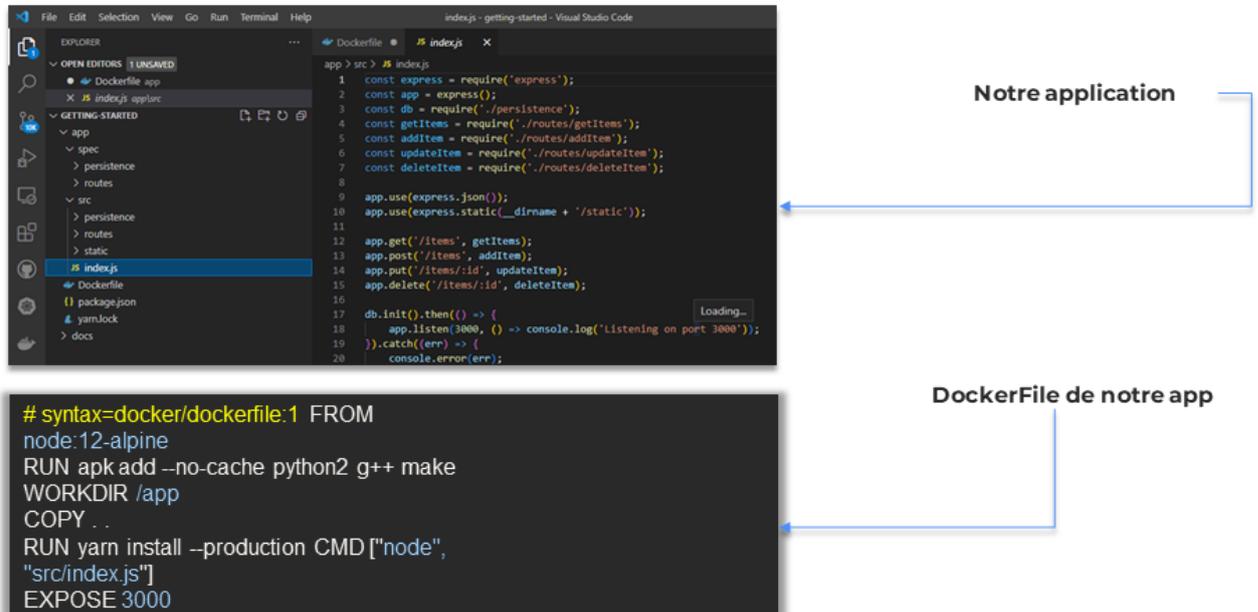


Figure 4.1: DockerFile

- (b) **Crée l'image du conteneur et Mettre l'image dans le registre docker**

```

$ docker build -t nom-de-l'image
$ docker build -t getting-started

```

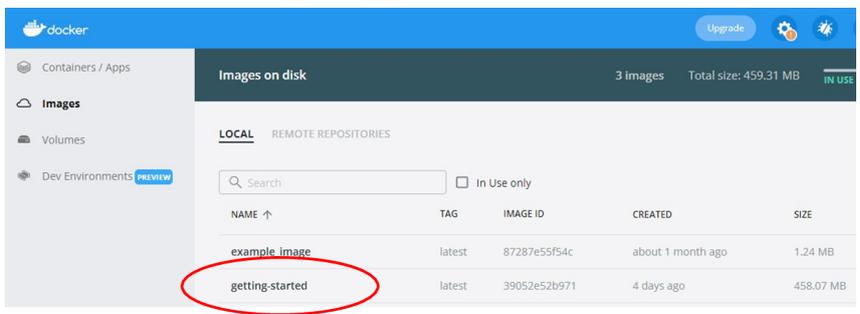
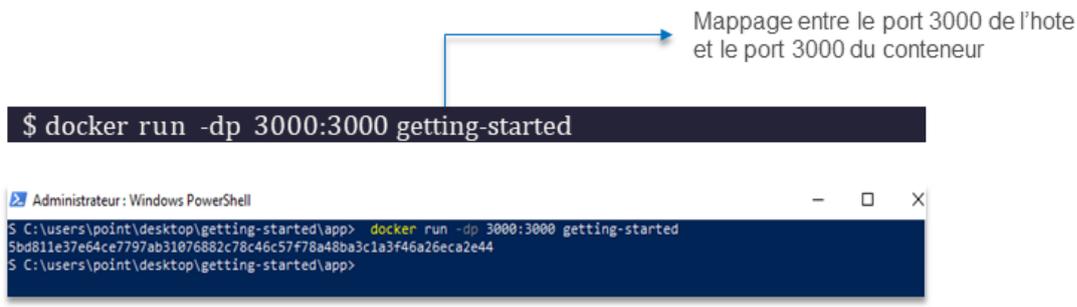


Figure 4.2: Création de l'image

- (c) **Deployer l'application avec docker:** Nous allons lancer un conteneur docker afin de tester le déploiement de notre application



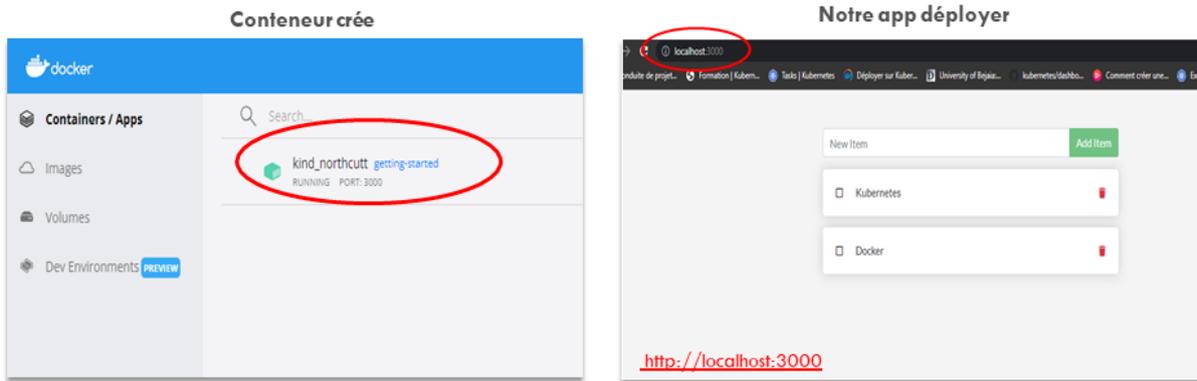
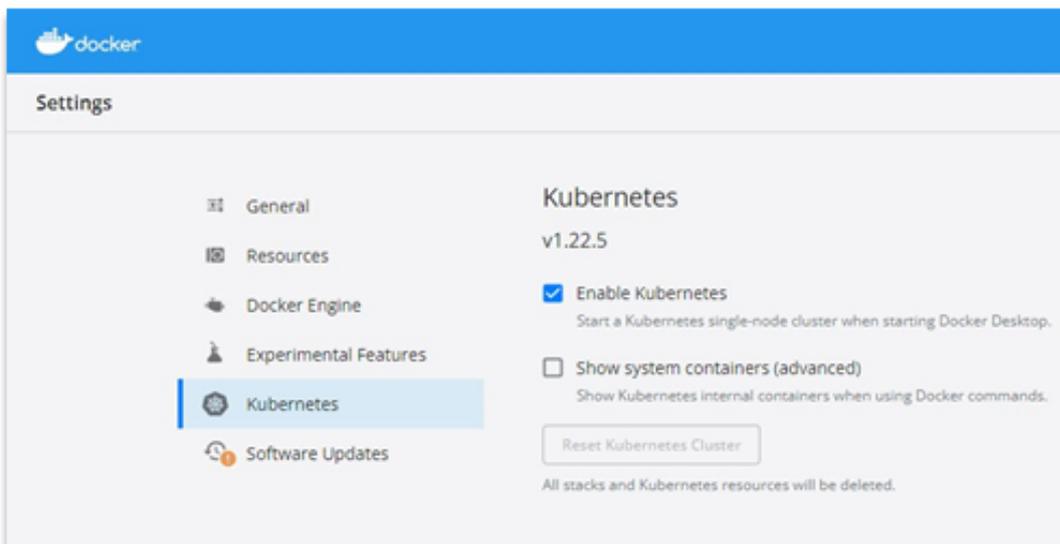


Figure 4.3: Déploiement avec Docker

- (d) **Déployer l'application avec kubernetes:** Nous devons tout d'abord lancer un cluster kubernetes pour ce faire, nous pouvons utiliser soit Minikube soit l'environnement kubernetes complet intégré dans docker desktop.

1. Kubernetes dans docker-desktop



2 Utiliser minikube

```
$ Minikube start
```

```
$ Minikube status
```

```
Administrateur: Windows PowerShell
PS C:\users\point\desktop\getting-started\app> minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Figure 4.4: Déploiement avec Kubernetes

Ensuite nous allons créer un fichier yaml qui décrit tous les composants et configurations de pour notre application.

Fichier yaml appelée (bb.yaml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bb-demo
  namespace: default
spec:
  replicas: 1
  containers:
  - name: bb-site
    image: getting-started
---
apiVersion: v1
kind: Service
metadata:
  name: bb-entypoint
  namespace: default
spec:
  type: NodePort
  selector:
    bb: web
  ports:
  - port: 3000
    targetPort: 3000
    nodePort: 30001
```



Partie deployment

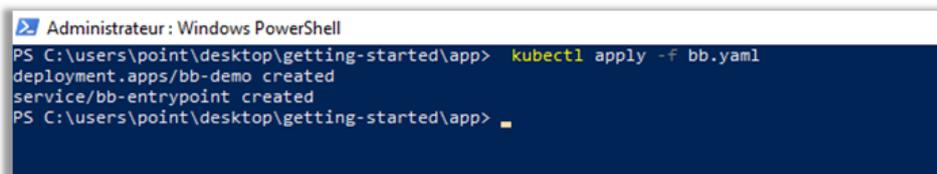
Partie service

Déployer notre application

```
$ kubectl apply -f bb.yaml
```

Indique que les objets kubernetes ont été crée

```
Administrateur: Windows PowerShell
PS C:\users\point\desktop\getting-started\app> kubectl apply -f bb.yaml
deployment.apps/bb-demo created
service/bb-entypoint created
PS C:\users\point\desktop\getting-started\app> _
```



```
$ kubectl get deployments
```

```
$ kubectl get services
```

Le déploiement et le service sont opérationnels

```
C:\users\point\desktop\getting-started\app> kubectl get deployments
E   READY   UP-TO-DATE   AVAILABLE   AGE
demo   0/1       1           0           8m7s
C:\users\point\desktop\getting-started\app> kubectl get services
E   TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
entypoint   NodePort     10.107.237.63 <none>         3000:30002/TCP  9m39s
ernetes     ClusterIP    10.96.0.1     <none>         443/TCP          2d3h
```

Figure 4.5: Déploiement avec Kubernetes

3. Erreur du déploiement

Nous avons rencontré l'erreur ImagePullBackoff qui signifie que Le pod n'arrive pas à extraire l'image du conteneur.

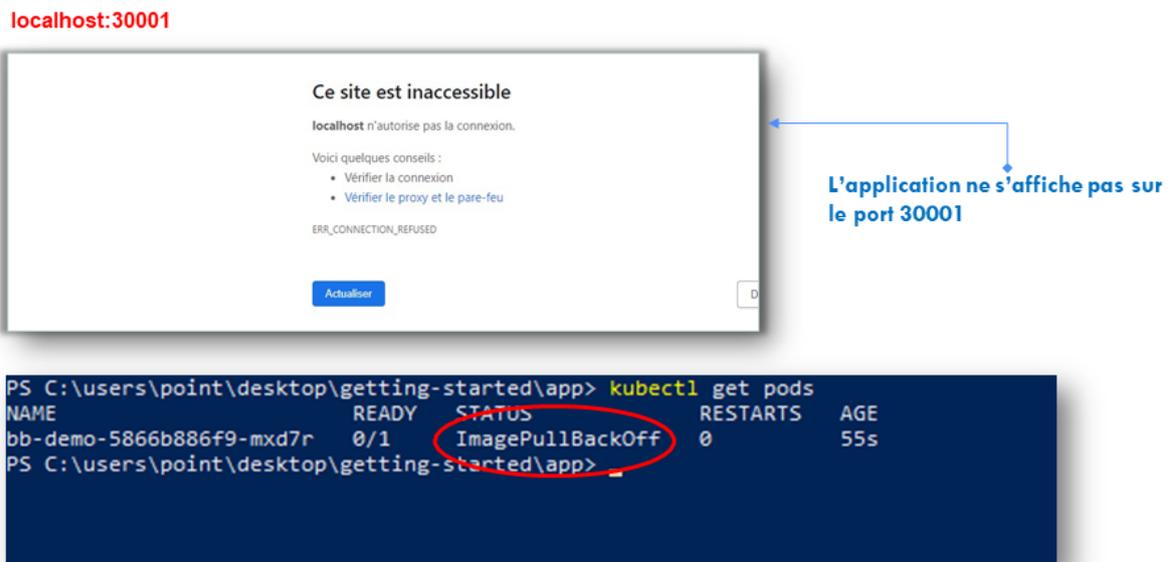


Figure 4.6: Erreur

4. Solutions

Dans le but de savoir si le problème provient de l'application. Nous avons testé ces deux images officielles : la simple image hello world et le serveur web open source Nginx

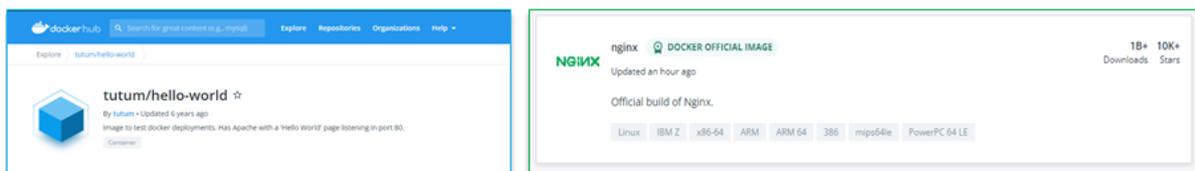


Figure 4.7: Deux images officiel

Nous avons remarqué la persistance du même problème ce qui nous a pousser à chercher plus loin pour en déduire que c'était un soucis de version et du container rutime.

Pour que les pongs puissent s'exécuter il est nécessaire d'installer un environnement d'exécution de conteneur tel que docker.

Le composant dockershim de Kubernetes permettait d'utiliser Docker comme runtime de conteneur, Mais Dockershim a été supprimer complètement de k8s depuis la version 1.20 Par conséquent, nous avons utilisé deux méthodes pour résoudre cette erreur.

i. Utiliser le runtime docker

- **Downgrade kubernetes:** Nous avons essayé d'utiliser une version inférieure à 1.20, Qui inclue dockershim.

```
X minikube start -p aged --kubernetes-version=v1.21.2
```

- **Docker-cri:** Est la solution pour installer et utiliser dockershim, nous avons donc essayé de l'installer conformément aux instructions de la documentation officielle.

Docker Engine

Note: These instructions assume that you are using the `cri-dockerd` adapter to integrate Docker Engine with Kubernetes.

1. On each of your nodes, install Docker for your Linux distribution as per [Install Docker Engine](#).
2. Install `cri-dockerd`, following the instructions in that source code repository.

For `cri-dockerd`, the CRI socket is `/run/cri-dockerd.sock` by default.

<https://kubernetes.io/docs/setup/production-environment/container-runtimes/#docker>

Build and install

To build this code (in a POSIX environment):

```
mkdir bin
cd src && go get && go build -o ../bin/cri-dockerd
```

To install, on a Linux system that uses systemd, and already has Docker Engine installed

```
# Run these commands as root
mkdir -p /usr/local/bin
install -o root -g root -m 0755 bin/cri-dockerd /usr/local/bin/cri-dockerd
cp -a packaging/systemd/* /etc/systemd/system
sed -i -e 's,./usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service
systemctl daemon-reload
systemctl enable cri-docker.service
systemctl enable --now cri-docker.socket
```

Figure 4.8: Docker Engine

Étant donné que la solution proposée est sous l'environnement Linux, nous avons essayé d'utiliser la distribution Linux sous Windows.

Install wsl + Ubuntu 18.04



```
Selection point@TGL: ~
** (process:7019): WARNING **: 18:05:20.000: Unable to register authentication agent: GDBus.Error:org.freedesktop.PolicyKit1.E
ror.Failed: Cannot determine user of subject
Error registering authentication agent: GDBus.Error:org.freedesktop.PolicyKit1.Error.Failed: Cannot determine user of subject
(polkit-error-quark, 0)
Failed to restart your-service-name.service: Interactive authentication required.
See system logs and 'systemctl status your-service-name.service' for details.
point@TGL:~$ clear
point@TGL:~$ # systemctl enable kubelet.service
point@TGL:~$ systemctl enable cri-docker.service
** (process:7026): WARNING **: 18:05:21.000: Unable to register authentication agent: GDBus.Error:org.freedesktop.PolicyKit1.E
ror.Failed: Cannot determine user of subject
Error registering authentication agent: GDBus.Error:org.freedesktop.PolicyKit1.Error.Failed: Cannot determine user of subject
(polkit-error-quark, 0)
Failed to enable unit: Interactive authentication required.
point@TGL:~$ systemctl daemon-reload
** (process:7031): WARNING **: 18:05:21.000: Unable to register authentication agent: GDBus.Error:org.freedesktop.PolicyKit1.E
ror.Failed: Cannot determine user of subject
Error registering authentication agent: GDBus.Error:org.freedesktop.PolicyKit1.Error.Failed: Cannot determine user of subject
(polkit-error-quark, 0)
Failed to reload daemon: Interactive authentication required.
point@TGL:~$
```

Erreur rencontrée:

WSL ne permet pas l'utilisation de la commande systemctl

Figure 4.9: Distribution linux sous windows

- ii. **Containerd:** Nous avons essayé de modifier le runtime CRI en choisissant containerd qui possède la plupart des fonctionnalités Docker pour l'exécution des conteneurs, la gestion de la mémoire et la gestion des images.

```
Administrateur: Invite de commandes
C:\WINDOWS\system32>containerd --version
containerd github.com/containerd/containerd v1.5.11 3df54a852345ae127d1fa3092b95168e4a88e2f8
C:\WINDOWS\system32>
```

```
Administrateur: Windows PowerShell
PS C:\> minikube start --driver=hyperv --container-runtime=containerd
* minikube v1.25.2 sur Microsoft Windows 10 Pro 10.0.19043 Build 19043
* Utilisation du pilote hyperv basé sur le profil existant
ns le cluster minikube
* Préparation de Kubernetes v1.23.3 sur containerd 1.4.12...
- kubelet.housekeeping-interval=5m
- Génération des certificats et des clés
- Démarrage du plan de contrôle ...
- Configuration des règles RBAC ...
* Configuration de bridge CNI (Container Networking Interface)...
* Vérification des composants Kubernetes...
- Utilisation de l'image gcr.io/k8s-minikube/storage-provisioner:v5
* Modules activés: storage-provisioner
* Terminé ! kubectl est maintenant configuré pour utiliser "minikube" cluster et espace de noms "default" par défaut.
PS C:\> kubectl get nodes
NAME STATUS ROLES AGE VERSION
minikube Ready control-plane,master 5m24s v1.23.3
PS C:\>
```

Figure 4.10: CRI containerd

iii. Continuer le déploiement avec kubernetes

Revenons aux deux précédentes images Hello world et Nginx que nous avons essayé d'implémenter.

Nous allons créer deux fichiers yaml pour chacune puis nous procédons à leur déploiement.

```
Help podyaml Untitled (Workspace) - Visual Studio Code
Extension: Kubernetes Support
app > podyaml > {} spec > [ ] ports > {} 0
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nginx
5 spec:
6   replicas: 2
7   selector:
8     matchLabels:
9       app: nginx
10  template:
11    metadata:
12      labels:
13        editor: vscode
14        app: nginx
15    spec:
16      containers:
17        - name: nginx
18          image: nginx:latest
19          ports:
20            - containerPort: 80
21  ---
22 kind: Service
23 apiVersion: v1
24 metadata:
25   name: nginx
26 spec:
27   selector:
28     app: nginx
29   type: NodePort
30   ports:
31     - port: 80
32       targetPort: 80
33
34
```

Pod.yaml

```
Help simpleyaml Untitled (Workspace) - Visual Studio Code
Extension: Kubernetes Support
app > simpleyaml > {} spec > [ ] ports > {} 0 # port
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: tutum-deployment
5 spec:
6   replicas: 1
7   selector:
8     matchLabels:
9       app: tutum
10  template:
11    metadata:
12      labels:
13        editor: vscode
14        app: tutum
15    spec:
16      containers:
17        - name: tutum
18          image: tutum/hello-world:latest
19          ports:
20            - containerPort: 80
21  ---
22 kind: Service
23 apiVersion: v1
24 metadata:
25   name: tutum-service
26 spec:
27   selector:
28     app: tutum
29   type: NodePort
30   ports:
31     - port: 5500
32       targetPort: 80
33
34
```

Simple.yaml

```
Kubectl apply -f pod.yaml
```

```
Kubectl get deployment
```

```
Kubectl get svc
```

```
Kubectl get pods
```

```
Kubectl port-forward service/nginx 80:80
```

Figure 4.11: Fichiers yaml des deux images et lancement du déploiement

```

Administrateur : Windows PowerShell
PS C:\Users\lyesd\desktop\app> docker images
REPOSITORY          TAG          IMAGE ID      CREATED       SIZE
nginx                latest      fa5269854a5e 2 weeks ago  142MB
hubproxy.docker.internal:5000/docker/desktop-kubernetes-apiserver  v1.22.5    059e6cd8cf78 4 months ago  128MB
k8s.gcr.io/kube-apiserver  v1.22.5    059e6cd8cf78 4 months ago  128MB
hubproxy.docker.internal:5000/docker/desktop-kubernetes-scheduler  v1.22.5    935d8fdc2d52 4 months ago  52.7MB
k8s.gcr.io/kube-scheduler  v1.22.5    935d8fdc2d52 4 months ago  52.7MB
k8s.gcr.io/kube-proxy      v1.22.5    8f8fdd6672d4 4 months ago  104MB
hubproxy.docker.internal:5000/docker/desktop-kubernetes-proxy      v1.22.5    8f8fdd6672d4 4 months ago  104MB
hubproxy.docker.internal:5000/docker/desktop-kubernetes-controller-manager  v1.22.5    04185bc88e08 4 months ago  122MB
k8s.gcr.io/kube-controller-manager  v1.22.5    04185bc88e08 4 months ago  122MB
docker/desktop-vpnkit-controller  v2.0       8c2c38aa676e 12 months ago  21MB
docker/desktop-storage-provisioner  v2.0       99f89471f470 12 months ago  41.9MB
k8s.gcr.io/pause          3.5       ed210e3e4a5b 13 months ago  683kB
hubproxy.docker.internal:5000/docker/desktop-kubernetes-pause      3.5       ed210e3e4a5b 13 months ago  683kB
k8s.gcr.io/coredns/coredns  v1.8.0    296a6d5035e2 18 months ago  42.5MB
k8s.gcr.io/etcd          3.4.13-0  0369cf4303ff 20 months ago  253MB
tutum/hello-world        latest     31e17b0746e4 6 years ago   17.8MB
PS C:\Users\lyesd\desktop\app> kubectl apply -f simple.yaml
deployment.apps/tutum-deployment created
service/tutum-service unchanged
PS C:\Users\lyesd\desktop\app> kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
hello-minikube 1/1     1             1           4h30m
nginx         2/2     2             2           46m
tutum-deployment 1/1     1             1           28s
PS C:\Users\lyesd\desktop\app> kubectl get service
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-minikube NodePort      10.108.179.194  <none>        8080:30455/TCP  4h30m
kubernetes    ClusterIP     10.96.0.1        <none>        443/TCP          4h56m
nginx         NodePort      10.102.10.188   <none>        5555:31064/TCP  46m
tutum-service NodePort      10.111.164.241 <none>        81:31591/TCP    3m38s
PS C:\Users\lyesd\desktop\app> kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
hello-minikube-7bc9d7884c-vn4sc 1/1     Running   0           4h30m
nginx-8587df58cf-25mpd          1/1     Running   0           46m
nginx-8587df58cf-d2nms          1/1     Running   0           46m
tutum-deployment-79686cd646-9c5jp 1/1     Running   0           47s
PS C:\Users\lyesd\desktop\app> minikube service tutum-service
PS C:\Users\lyesd\desktop\app> minikube service tutum-service --url
http://172.22.68.177:31591
PS C:\Users\lyesd\desktop\app> kubectl port-forward service/tutum-service 81:8080
error: Service tutum-service does not have a service port 8080
PS C:\Users\lyesd\desktop\app> kubectl port-forward service/tutum-service 8080:81
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::]:8080 -> 80
Handling connection for 8080
PS C:\Users\lyesd\desktop\app> kubectl apply -f simple.yaml
deployment.apps/tutum-deployment unchanged
service/tutum-service configured
PS C:\Users\lyesd\desktop\app> kubectl port-forward service/tutum-service 8080:5500
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::]:8080 -> 80
Handling connection for 8080
PS C:\Users\lyesd\desktop\app> kubectl port-forward service/tutum-service 3000:5500
Forwarding from 127.0.0.1:3000 -> 80
Forwarding from [::]:3000 -> 80
Handling connection for 3000

```

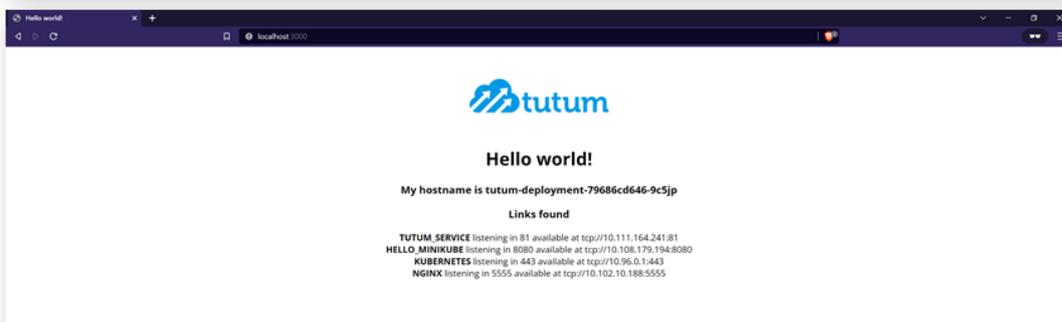
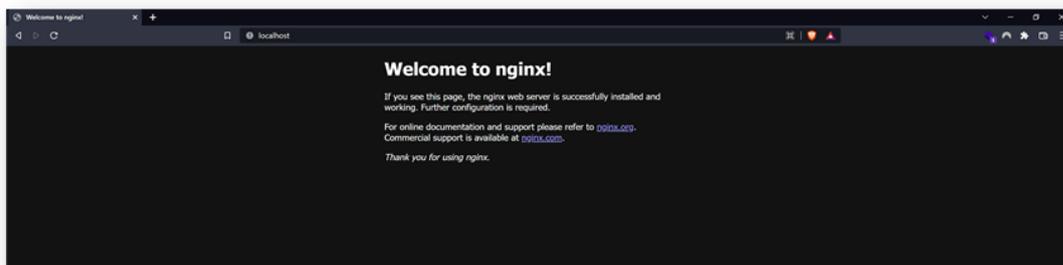
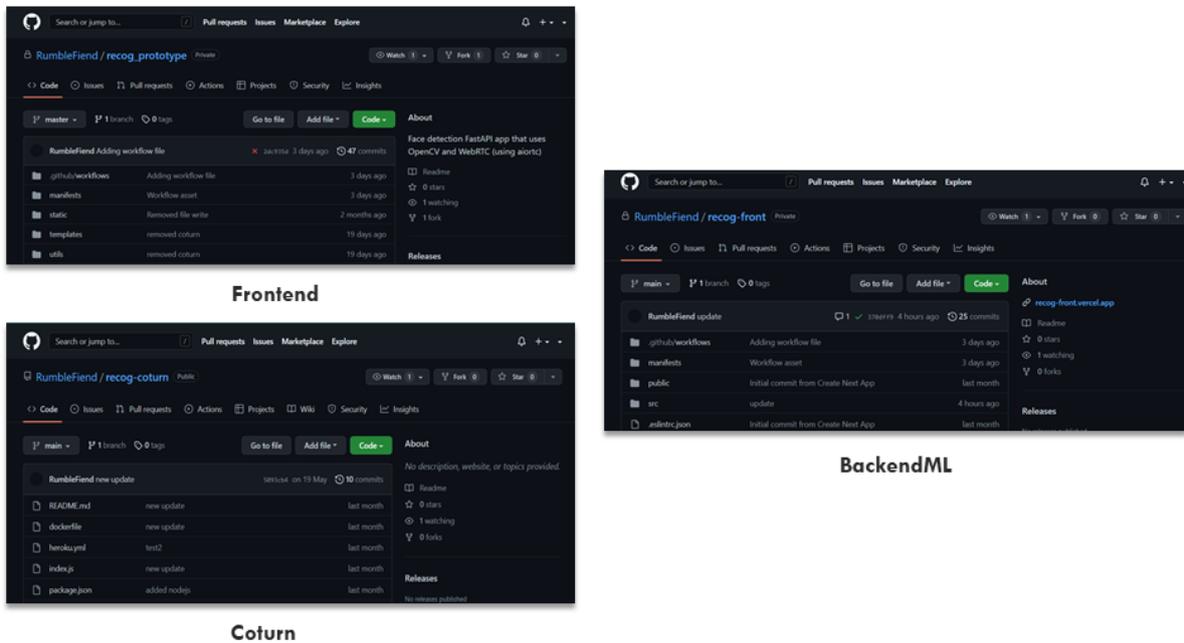


Figure 4.12: Resultat

5. **Déploiement du système avec Minikube:** La solution cloud computing Intelligent pour la reconnaissance en temps Réel dispose d'une architecture microservices composée de trois parties : Frontend, Coturn et Backend ML. Nous mettrons donc en place trois services.



[29] [30] [31]

Figure 4.13: les repos git

- **Déploiement du front-end:** Nous allons passer par les mêmes étapes précédentes, c'est-à-dire que nous allons construire notre image puis la taguer pour l'inclure dans le registre docker, puis nous configurons le fichier yaml et déployons l'image du frontend.

Tag and push notre image sur docker hub dans le repo « po001int »

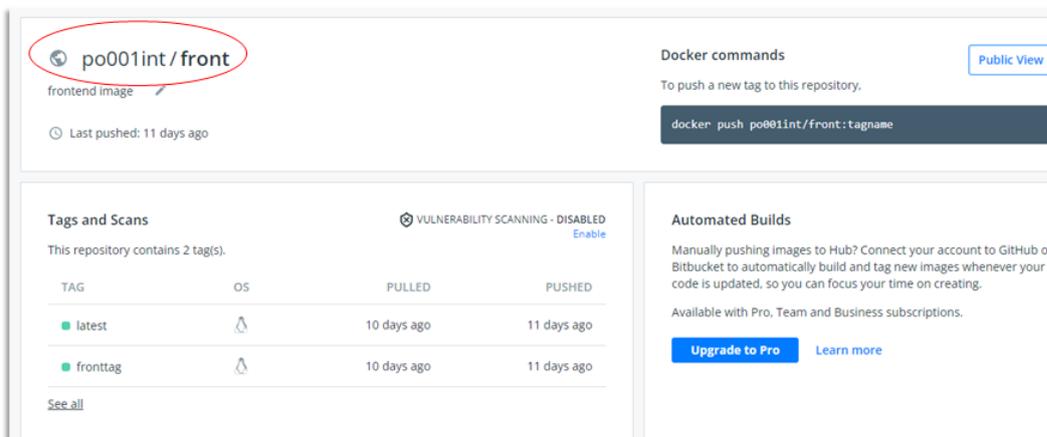


Figure 4.14: Image de l'application

Avec yaml manuellement

```
Administrateur: Windows PowerShell
PS C:\users\point\desktop\front-recog> kubectl apply -f front.yaml
deployment.apps/front unchanged
service/front unchanged
PS C:\users\point\desktop\front-recog> kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
front     1/1     1             1           7m40s
PS C:\users\point\desktop\front-recog> kubectl get svc
NAME      TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
front     NodePort      10.111.234.19 <none>        5555:32702/TCP   7m43s
kubernetes ClusterIP  10.96.0.1     <none>        443/TCP          3h2m
PS C:\users\point\desktop\front-recog> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
front-76ff5dc9c6-ws1sn              1/1     Running   0           7m47s
PS C:\users\point\desktop\front-recog> kubectl port-forward service/front 3000:5555
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

Avec CLI

```
PS C:\users\point\desktop\front-recog> kubectl create deployment front --image=po001int/front
deployment.apps/front created
PS C:\users\point\desktop\front-recog> kubectl expose deployment front --type=NodePort --port=3000
service/front exposed
PS C:\users\point\desktop\front-recog> kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
front     1/1     1             1           53s
PS C:\users\point\desktop\front-recog> kubectl get svc
NAME      TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
front     NodePort      10.103.2.26  <none>        3000:32030/TCP   15s
kubernetes ClusterIP  10.96.0.1     <none>        443/TCP          3h6m
PS C:\users\point\desktop\front-recog> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
front-88574f45f-176nz              1/1     Running   0           64s
PS C:\users\point\desktop\front-recog> kubectl port-forward service/front 3000:5555
error: Service front does not have a service port 5555
PS C:\users\point\desktop\front-recog> kubectl port-forward service/front 5555:3000
Forwarding from 127.0.0.1:5555 -> 3000
Forwarding from [::1]:5555 -> 3000
Handling connection for 5555
Handling connection for 5555
Handling connection for 5555
PS C:\users\point\desktop\front-recog>
```

Résultat

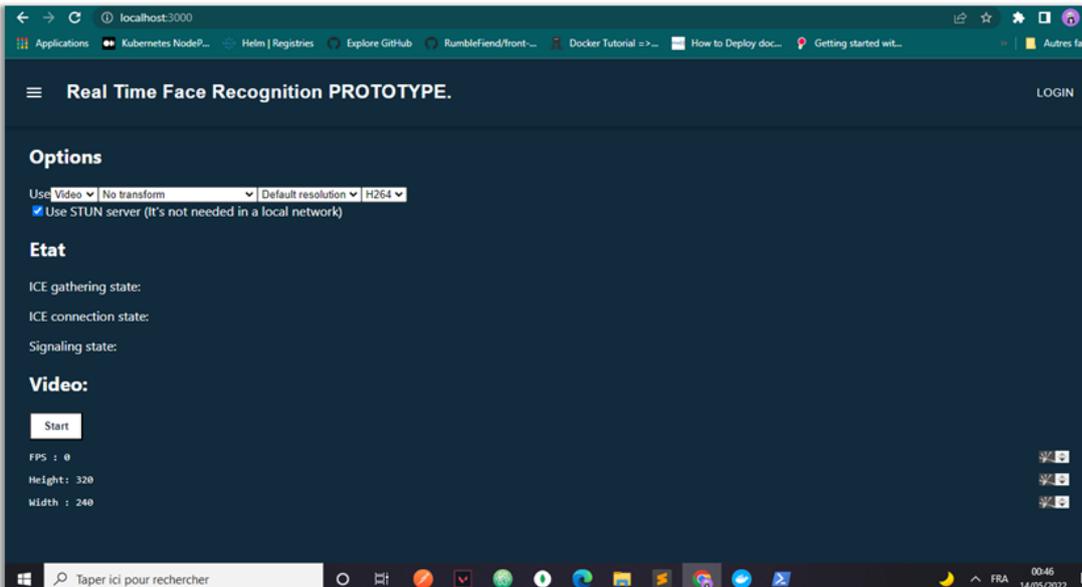


Figure 4.15: Image de l'application

- Déploiement du Coturn

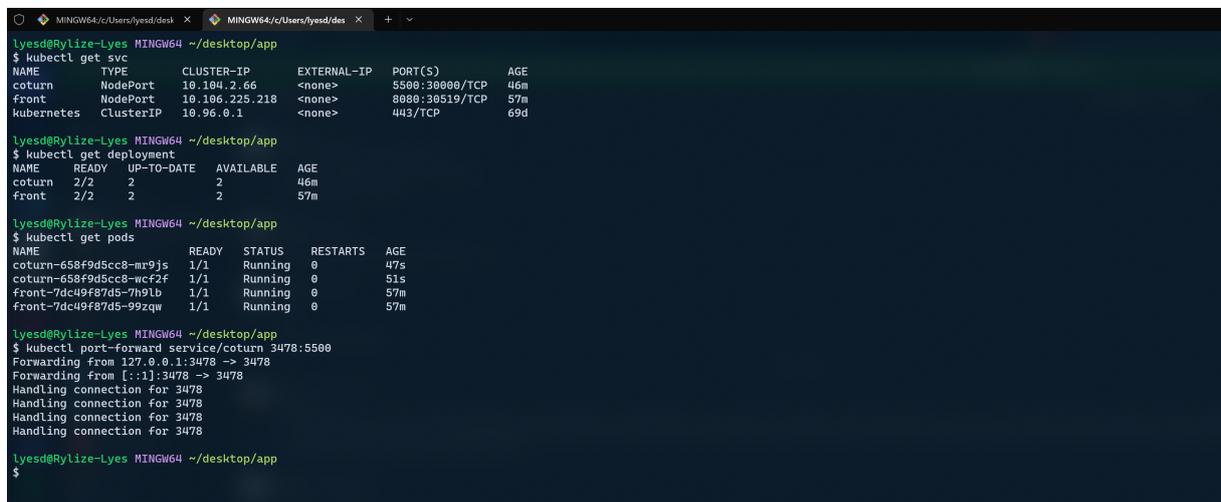
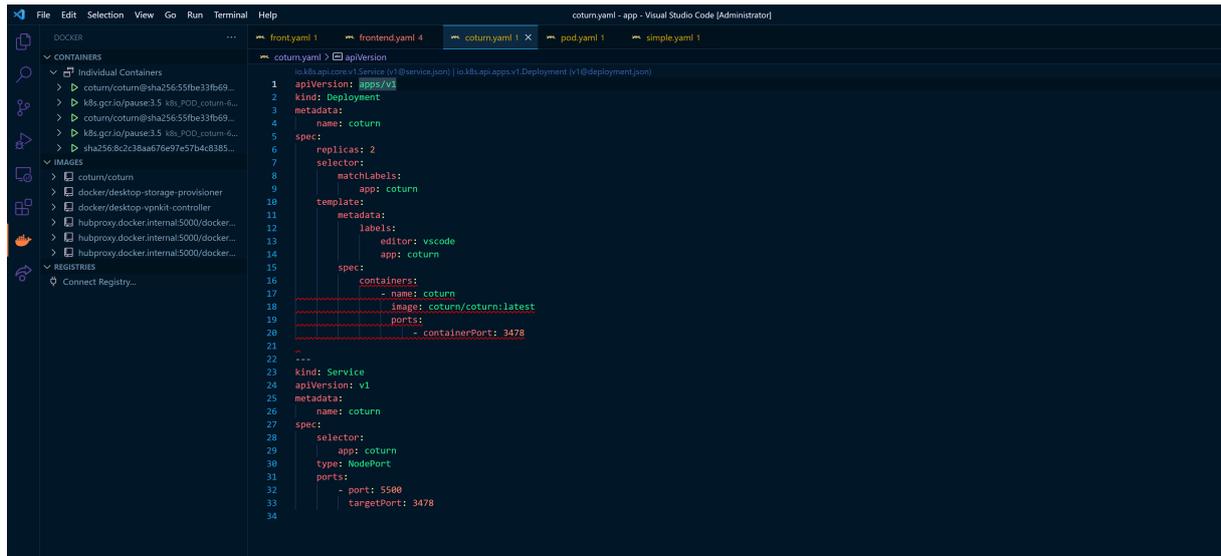


Figure 4.16: Coturn

6. Déploiement du BackendML

c'est là que nous utiliserons kubeflow, pour cela nous utiliserons MicroK8 qui est un K8 léger et concentré.

Avant de commencer, nous devons considérer les prérequis suivants : Ubuntu 20.04 ou version ultérieure, au moins 16Go de mémoire libre et 20Go d'espace disque.

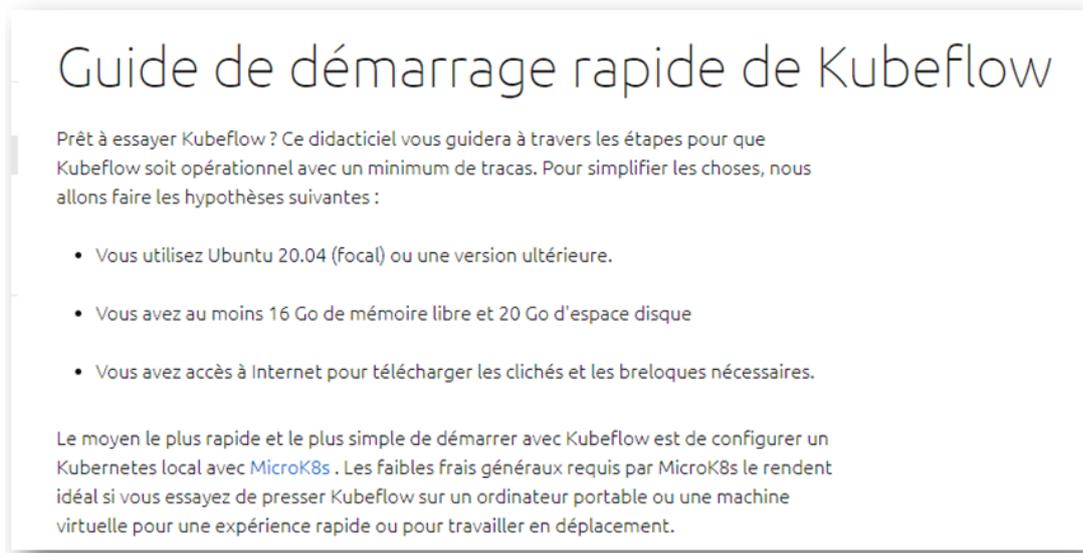


Figure 4.17: Kubeflow prérequis.

Pour cela, nous utiliserons une session d'un vps (virtual private server) qu'on a pu se procurer et qui est doté d'une capacité conforme aux exigences.

```
ssh ubuntu@195.15.243.101
```

```
ubuntu@k8e-test: ~
point@TGL MINGW64 ~ (master)
$ ssh ubuntu@195.15.243.101
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-27-generic x86_64)

Installer Microk8s

sudo snap install microk8s --classic --channel=1.21/stable

sudo usermod -a -G microk8s $USER
newgrp microk8s

sudo chown -f -R $USER ~/.kube

microk8s enable dns storage ingress metallb:195.15.243.101-195.15.243.101

microk8s status --wait-ready
```

Figure 4.18: vps et microk8s

```

ubuntu@k8s-test:~$ microk8s status --wait-ready
microk8s is running
high-availability: no
datastore master nodes: 127.0.0.1:19001
datastore standby nodes: none
addons:
  enabled:
    ha-cluster # Configure high availability on the current node
  disabled:
    ambassador # Ambassador API Gateway and Ingress
    cilium # SDN, fast with full network policy
    dashboard # The Kubernetes dashboard
    dns # CoreDNS
    fluentd # Elasticsearch-Fluentd-Kibana logging and monitoring
    gpu # Automatic enablement of Nvidia CUDA
    helm # Helm 2 - the package manager for Kubernetes
    helm3 # Helm 3 - Kubernetes package manager
    host-access # Allow Pods connecting to Host services smoothly
    ingress # Ingress controller for external access
    istio # Core Istio service mesh services
    jaeger # Kubernetes Jaeger operator with its simple config
    keda # Kubernetes-based Event Driven Autoscaling
    knative # The Knative framework on Kubernetes.
    kubeflow # Kubeflow for easy ML deployments
    linkerd # Linkerd is a service mesh for Kubernetes and other frameworks
    metallb # Loadbalancer for your Kubernetes cluster
    metrics-server # K8s Metrics Server for API access to service metrics
    multus # Multus CNI enables attaching multiple network interfaces to pods
    openebs # OpenEBS is the open-source storage solution for Kubernetes
    openfaas # openfaas serverless framework
    portainer # Portainer UI for your Kubernetes cluster
    prometheus # Prometheus operator for monitoring and logging
    rbac # Role-Based Access Control for authorisation
    registry # Private image registry exposed on localhost:32000
    storage # Storage class; allocates storage from host directory
    traefik # traefik Ingress controller for external access
ubuntu@k8s-test:~$

```

```

ubuntu@k8s-test:~$
- deploy application kubeflow-roles from charm-hub with 1 unit with latest/stable
- set annotations for kubeflow-roles
- upload charm kubeflow-volumes from charm-hub from channel latest/stable with architecture=amd64
- deploy application kubeflow-volumes from charm-hub with 1 unit with latest/stable
  added resource oci-image
- set annotations for kubeflow-volumes
- upload charm metacontroller-operator from charm-hub from channel latest/stable with architecture=amd64
- deploy application metacontroller-operator from charm-hub with 1 unit with latest/stable
  set annotations for metacontroller-operator
- upload charm minio from charm-hub from channel latest/stable with architecture=amd64
- deploy application minio from charm-hub with 1 unit with latest/stable
  added resource oci-image
- set annotations for minio
- upload charm m1md from charm-hub from channel latest/stable with architecture=amd64
- deploy application m1md from charm-hub with 1 unit with latest/stable
  added resource oci-image
- set annotations for m1md
- upload charm oidc-gatekeeper from charm-hub from channel latest/stable with architecture=amd64
- deploy application oidc-gatekeeper from charm-hub with 1 unit with latest/stable
  added resource oci-image
- set annotations for oidc-gatekeeper
- upload charm seldon-core from charm-hub from channel latest/stable with architecture=amd64
- deploy application seldon-controller-manager from charm-hub with 1 unit with latest/stable using seldon-core
  added resource oci-image
- set annotations for seldon-controller-manager
- upload charm training-operator from charm-hub from channel latest/stable with architecture=amd64
- deploy application training-operator from charm-hub with 1 unit with latest/stable
  added resource training-operator-image
- set annotations for training-operator
- add relation argo-controller - minio
- add relation dex-auth:oidc-client - oidc-gatekeeper:oidc-client
- add relation istio-pilot:ingress - dex-auth:ingress
- add relation istio-pilot:ingress - jupyter-ui:ingress
- add relation istio-pilot:ingress - kfp-ui:ingress
- add relation istio-pilot:ingress - kubeflow-dashboard:ingress
- add relation istio-pilot:ingress - kubeflow-volumes:ingress
- add relation istio-pilot:ingress - oidc-gatekeeper:ingress
- add relation istio-pilot:ingress-auth - oidc-gatekeeper:ingress-auth
- add relation istio-pilot:istio-pilot - istio-ingressgateway:istio-pilot
- add relation kfp-api - kfp-db
- add relation kfp-api:kfp-api - kfp-persistence:kfp-api
- add relation kfp-api:kfp-api - kfp-ui:kfp-api
- add relation kfp-api:kfp-viz - kfp-viz:kfp-viz
- add relation kfp-api:object-storage - minio:object-storage
- add relation kfp-profile-controller:object-storage - minio:object-storage
- add relation kfp-ui:object-storage - minio:object-storage
- add relation kubeflow-profiles - kubeflow-dashboard
- add relation m1md:grpc - envoy:grpc
Deploy of bundle completed.
ubuntu@k8s-test:~$

```

Figure 4.19: Installation de microk8s

Installer Juju

Utiliser pour déployer et gérer les composants qui composent Kubeflow.

```
sudo snap install juju --classic
```

```
juju bootstrap microk8s
```

```
juju add-model kubeflow
```

Déployer kubeflow

```
juju deploy kubeflow-lite --trust
```

```
watch -c juju status --color
```

Configuration

```
juju config dex-auth static-username=admin
```

```
juju config dex-auth static-password=admin
```

Access to Dashboard

```
exit
```

```
ssh -D 9999 ubuntu@195.15.243.101
```

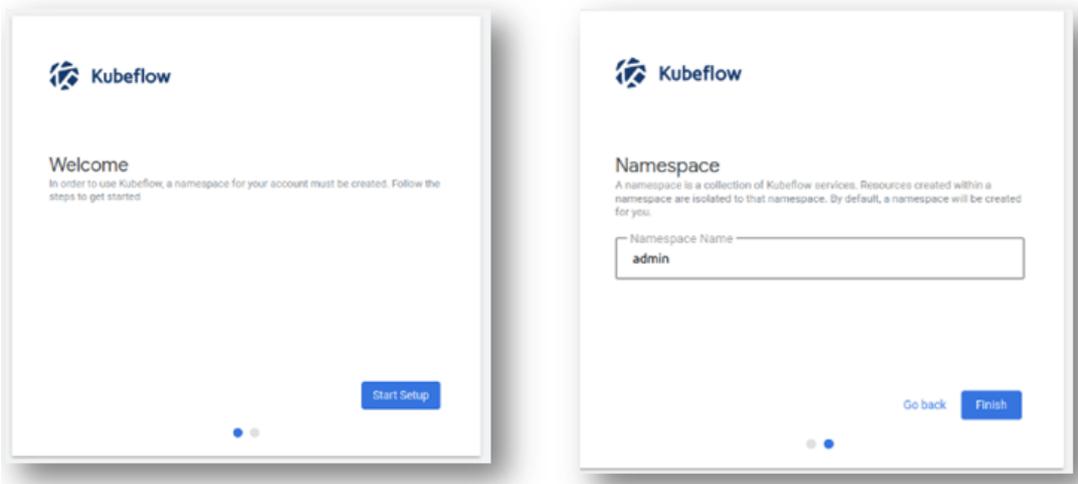


Figure 4.20: Déploiement de kubeflow

http://1195.15.243.101.nip.io

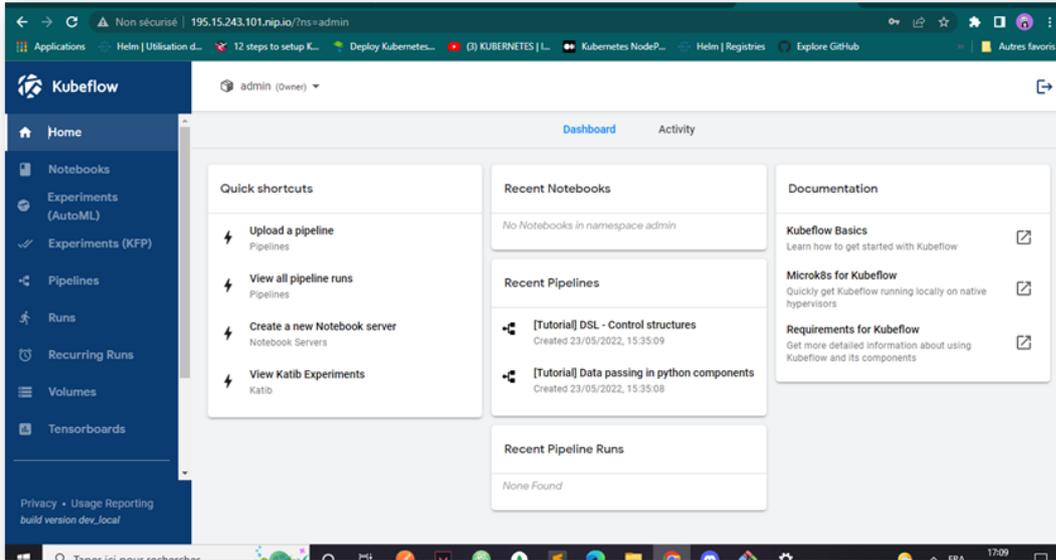


Figure 4.21: Interface kubeflow

4.3 Déploiement avec trois nœuds physiques

Pour créer un cluster k8s multi-nœuds, nous utiliserons kubeadm. Dans notre cas, nous allons opérer sur trois nœuds: Master node, Worker1 et Worker2.

Installer et configurer kubeadm

```
sudo apt update
```

```
sudo systemctl --system
```

```
sudo apt install -y docker docker.io containerd
```

```
sudo usermod -aG docker $USER
```

```
sudo newgrp docker
```

```
echo "configuring docker to use systemd" << EOF | sudo tee /etc/docker/daemon.json { "exec-opts": ["native.cgroupdriver=systemd"] } EOF
```

```
" Restart docker " sudo systemctl daemon-reload sudo systemctl restart docker
```

```
" Restart containerd " sudo systemctl restart containerd sudo systemctl enable containerd sudo systemctl enable docker
```

```
" Installing kubernetes dependencies " sudo apt install -y apt-transport-https curl curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

```
echo "adding apt repository for kubernetes ..." cat << EOF | sudo tee /etc/apt/sources.list.d/kubernetes.list deb https://apt.kubernetes.io/ kubernetes-xenial main EOF-y apt-transport-https curl
```

```
sudo apt update
```

Figure 4.22: Installation et configuration de Kubeadm

```
" Installing kubeadm kubelet & kubectl ..."
sudo apt install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

```
sudo swapoff -a
cat << EOF | sudo tee -a $HOME/.bashrc
sudo swapoff -a EOF
```

```
" Restarting kubelet ..."
sudo systemctl daemon-reload
sudo systemctl restart kubelet
sudo systemctl enable kubelet
```

Figure 4.23: Installation et configuration de Kubeadm

Nous allons initialiser un cluster sur le master par la suite nous joindrons les deux workers.

```
master-node@master-node:~$ sudo kubeadm init --pod-network-cidr=10.0.0.0/16
[init] Using Kubernetes version: v1.24.2
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
[certs] Using certificateDir folder '/etc/kubernetes/pki'
[certs] Generating 'ca' certificate and key
[certs] Generating 'apiserver' certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes.kubernetes.default.kubernetes.default.svc.cluster.local master-node] and IPs [10.96.0.1 10.2.12.217]
[certs] Generating 'apiserver-kubelet-client' certificate and key
[certs] Generating 'front-proxy-ca' certificate and key
[certs] Generating 'front-proxy-client' certificates and key
[certs] Generating 'etcd/ca' certificate and key
[certs] Generating 'etcd/peer' certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost master-node] and IPs [10.2.12.217 127.0.0.1 ::1]
[certs] Generating 'etcd/peer' certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost master-node] and IPs [10.2.12.217 127.0.0.1 ::1]
[certs] Generating 'etcd/healthcheck-client' certificate and key
[certs] Generating 'apiserver-etcd-client' certificate and key
[certs] Generating 'sa' key and public key
[kubeconfig] Using kubeconfig folder '/etc/kubernetes'
[kubeconfig] Writing 'admin.conf' kubeconfig file
[kubeconfig] Writing 'kubelet.conf' kubeconfig file
[kubeconfig] Writing 'controller-manager.conf' kubeconfig file
[kubeconfig] Writing 'scheduler.conf' kubeconfig file
[kubelet-start] Writing kubelet environment file with flags to file '/var/lib/kubelet/kubeadm-flags.env'
[kubelet-start] Writing kubelet configuration to file '/var/lib/kubelet/config.yaml'
[kubelet-start] Starting the kubelet
[control-plane] Using manifest folder '/etc/kubernetes/manifests'
[control-plane] Creating static Pod manifest for 'kube-apiserver'
[control-plane] Creating static Pod manifest for 'kube-controller-manager'
[control-plane] Creating static pod manifest for 'kube-scheduler'
[etcd] Creating static Pod manifest for local etcd in '/etc/kubernetes/manifests'
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory '/etc/kubernetes/manifests'. This can take up to 4m5s
[apiclient] All control plane components are healthy after 17.50174 seconds
[upload-config] Storing the configuration used in ConfigMap 'kubeadm-config' in the 'kube-system' Namespace
[kubelet] Creating a ConfigMap 'kubelet-config' in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node master-node as control-plane by adding the labels [node-role.kubernetes.io/control-plane:kubernetes.io/exclude-from-external-load-balancers]
[mark-control-plane] Marking the node master-node as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: j1njbh_mppm9jrxrkxhlls
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the 'cluster-info' ConfigMap in the 'kube-public' namespace
[kubelet-finalize] Updating '/etc/kubernetes/kubelet.conf' to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:
```

Figure 4.24: Kubeadm

```

master-node@master-node:~$
[certs] etcd/server serving cert is signed for DNS names [localhost master-node] and IPs [10.2.12.217 127.0.0.1 ::1]
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost master-node] and IPs [10.2.12.217 127.0.0.1 ::1]
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Starting the kubelet
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"
[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s
[apiclient] All control plane components are healthy after 17.501734 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node master-node as control-plane by adding the labels: [node-role.kubernetes.io/master:NoSchedule node-role.kubernetes.io/exclude-from-external-load-balancers]
[mark-control-plane] Marking the node master-node as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule node-role.kubernetes.io/control-plane:NoSchedule]
[bootstrap-token] Using token: j1njbh.mppm0jrfxkhehlis
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public" namespace
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rotatable kubelet client certificate and key
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run 'kubectl apply -f [podnetwork].yaml' with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.2.12.217:6443 --token j1njbh.mppm0jrfxkhehlis \
--discovery-token-ca-cert-hash sha256:a7f963014fef5ad099260d4ee30e1c02a44f82d0c67a40296c5db264e698a51

master-node@master-node:~$ sudo kubectl apply -f calico.yaml
error: error loading config file "/etc/kubernetes/admin.conf": open /etc/kubernetes/admin.conf: permission denied
master-node@master-node:~$ sudo kubectl apply -f calico.yaml
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/calicoconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamlocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
service/calico-typha created
deployment.apps/calico-typha created
poddisruptionbudget.policy/calico-typha created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
poddisruptionbudget.policy/calico-kube-controllers created
master-node@master-node:~$ sudo kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
master-node         NotReady control-plane 14m   v1.24.0
worker2-node        NotReady     9m26s v1.24.0
master-node@master-node:~$

worker2-node@worker2-node:~$ kubeadm join 10.2.12.217:6443 --token j1njbh.mppm0jrfxkhehlis \
--discovery-token-ca-cert-hash sha256:a7f963014fef5ad099260d4ee30e1c02a44f82d0c67a40296c5db264e698a51
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
[ERROR] IPPrivilegedUser: user is not running as root
[preflight] If you know what you are doing, you can make a check non-fatal with '--ignore-preflight-errors=...'
To see the stack trace of this error execute with '--v=5 or higher
worker2-node@worker2-node:~$ kubeadm join 10.2.12.217:6443 --token j1njbh.mppm0jrfxkhehlis --discovery-token-ca-cert-hash sha256:a7f963014fef5ad099260d4ee30e1c02a44f82d0c67a40296c5db264e698a51
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
[ERROR] IPPrivilegedUser: user is not running as root
[preflight] If you know what you are doing, you can make a check non-fatal with '--ignore-preflight-errors=...'
To see the stack trace of this error execute with '--v=5 or higher
worker2-node@worker2-node:~$ sudo kubeadm join 10.2.12.217:6443 --token j1njbh.mppm0jrfxkhehlis --discovery-token-ca-cert-hash sha256:a7f963014fef5ad099260d4ee30e1c02a44f82d0c67a40296c5db264e698a51
[sudo] password for worker2-node:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

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

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
worker2-node@worker2-node:~$

```

Figure 4.25: Kubeadm

Nous aurons besoin de Helm, le gestionnaire de packages pour Kubernetes dans le but d'ajouter des packages nécessaires. dans notre cas nous allons déployer avec le dashboard k8s.

1- Installation de Helm

```
choco install kubernetes-helm
```

2- Initialiser un dépôt de charts Helm

```
helm repo add stable https://charts.helm.sh/stable
```

3- Installer un package pour tester

```
helm install happy-panda bitnami/wordpress
```



les dépôts publics de charts Helm sont dans [ArtifactHUB](#)

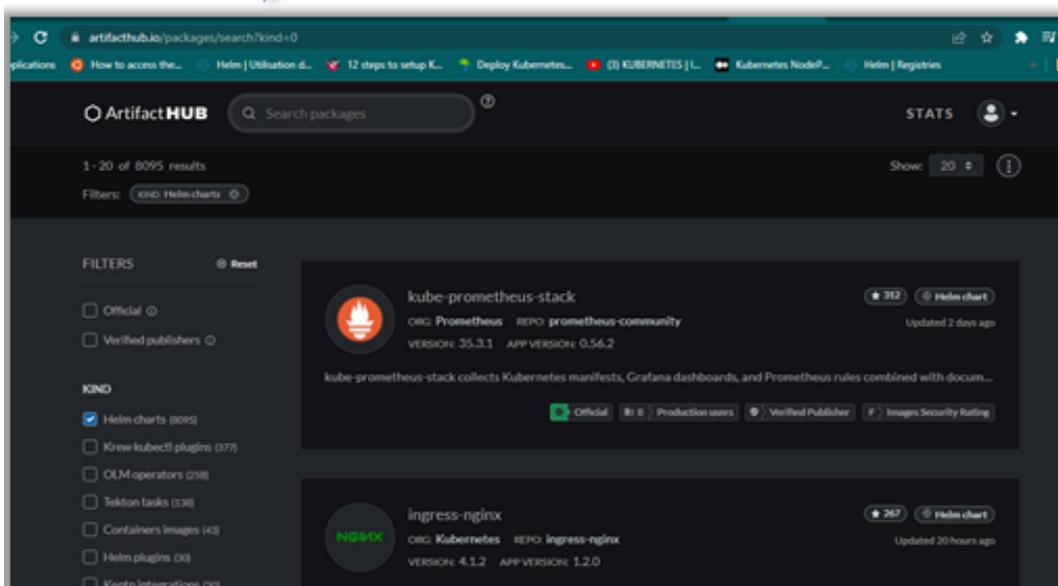


Figure 4.26: Helm

```

File Edit View Terminal Tabs Help
Get:2 http://es.archive.ubuntu.com/ubuntu bionic-updates InRelease [88,7 kB]
Hit:3 http://ppa.launchpad.net/x2go/stable/ubuntu bionic InRelease
Get:4 http://security.ubuntu.com/ubuntu bionic-security InRelease [88,7 kB]
Get:5 https://baltoedn.com/helm/stable/debian all InRelease [7.652 B]
Get:6 https://baltoedn.com/helm/stable/debian all/main amd64 Packages [2.684 B]
Get:7 http://es.archive.ubuntu.com/ubuntu bionic-backports InRelease [74,6 kB]
Get:8 http://security.ubuntu.com/ubuntu bionic-security/main amd64 DEP-11 Metadata [55,1 kB]
Hit:9 https://deb.releases.teleport.dev stable InRelease
Hit:10 https://ngrok-agent.s3.amazonaws.com buster InRelease
Get:12 http://es.archive.ubuntu.com/ubuntu bionic-updates/main amd64 DEP-11 Metadata [297 kB]
Hit:13 https://linux.teamviewer.com/deb stable InRelease
Get:14 http://es.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 DEP-11 Metadata [302 kB]
0% [Connected to packages.cloud.google.com (142.251.209.46)] [14 Components-amd64 66,3 kB/302 kB 22%] [Waiting for headers]
Get:15 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 DEP-11 Metadata [61,0 kB]
Get:16 http://es.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 DEP-11 Metadata [2.468 B]
Hit:11 https://packages.cloud.google.com/apt/kubernetes-xenial InRelease
Get:17 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 DEP-11 Metadata [2.464 B]
Get:18 http://es.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 DEP-11 Metadata [9.276 B]
Fetched 991 kB in 13s (74,2 kB/s)
Reading package lists... Done
master-node@master-node:~$ sudo apt-get install helm
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  golang-1.10-go golang-1.10-race-detector-runtime golang-1.10-src golang-race-detector-runtime golang-src
  linux-hwe-5.4-headers-5.4.0-110 pkg-config
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  helm
0 upgraded, 1 newly installed, 0 to remove and 133 not upgraded.
Need to get 13,9 MB of archives.
After this operation, 46,2 MB of additional disk space will be used.
Get:1 https://baltoedn.com/helm/stable/debian all/main amd64 helm amd64 3.9.0-1 [13,9 MB]
Fetched 13,9 MB in 15s (953 kB/s)
Selecting previously unselected package helm.
(Reading database ... 231988 files and directories currently installed.)
Preparing to unpack .../helm_3.9.0-1_amd64.deb ...
Unpacking helm (3.9.0-1) ...
Setting up helm (3.9.0-1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
master-node@master-node:~$ sudo apt-get install helm
Reading package lists... Done
Building dependency tree
Reading state information... Done
helm is already the newest version (3.9.0-1).
The following packages were automatically installed and are no longer required:
  golang-1.10-go golang-1.10-race-detector-runtime golang-1.10-src golang-race-detector-runtime golang-src linux-hwe-5.4-headers-5.4.0-110 pkg-config
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 133 not upgraded.
master-node@master-node:~$ c

```

```

# Add kubernetes-dashboard repository
helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
# Deploy a Helm Release named "kubernetes-dashboard" using the kubernetes-dashboard chart
helm install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard

```

```

helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
helm install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard

```

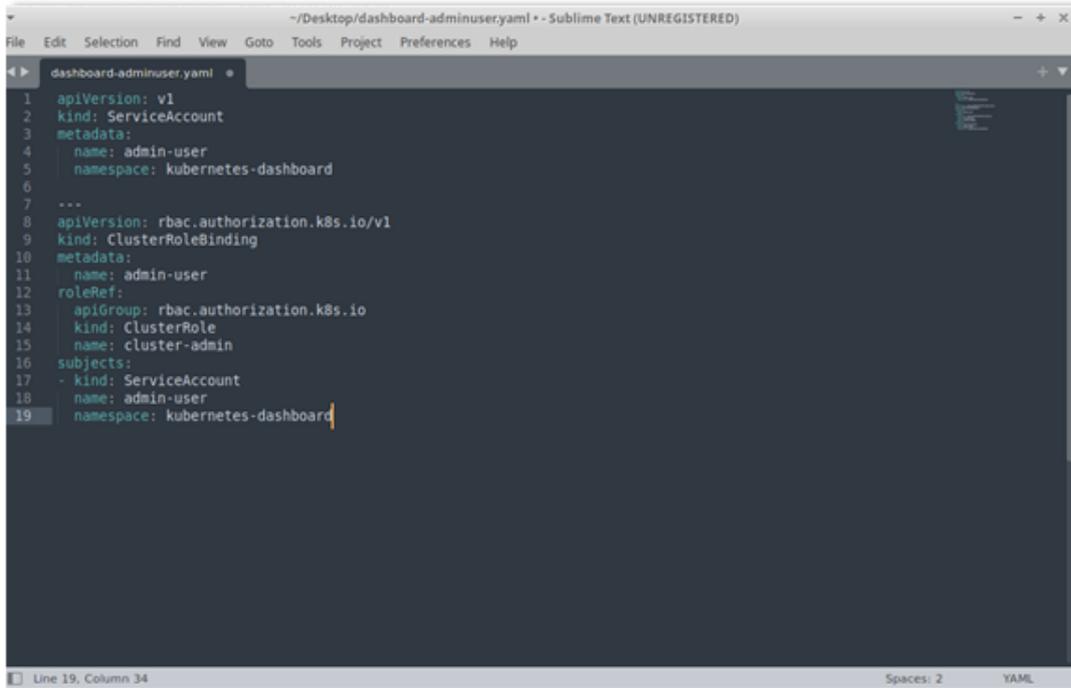
```

Terminal - master-node@master-node:~
File Edit View Terminal Tabs Help
master-node@master-node:~$ helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
"kubernetes-dashboard" already exists with the same configuration, skipping
master-node@master-node:~$ helm install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard
NAME: kubernetes-dashboard
LAST DEPLOYED: Mon Jun 20 12:10:36 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
*****
*** PLEASE BE PATIENT: kubernetes-dashboard may take a few minutes to install ***
*****
Get the Kubernetes Dashboard URL by running:
export POD_NAME=$(kubectl get pods -n default -l "app.kubernetes.io/name=kubernetes-dashboard,app.kubernetes.io/instance=kubernetes-dashboard" -o jsonpath="{.items[0].metadata.name}")
echo https://127.0.0.1:8443/
kubectl -n default port-forward $POD_NAME 8443:8443
master-node@master-node:~$
master-node@master-node:~$ helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
"kubernetes-dashboard" already exists with the same configuration, skipping
master-node@master-node:~$ export POD_NAME=$(kubectl get pods -n default -l "app.kubernetes.io/name=kubernetes-dashboard,app.kubernetes.io/instance=kubernetes-dashboard" -o jsonpath="{.items[0].metadata.name}")
master-node@master-node:~$ export POD_NAME=$(kubectl get pods -n default -l "app.kubernetes.io/name=kubernetes-dashboard,app.kubernetes.io/instance=kubernetes-dashboard" -o jsonpath="{.items[0].metadata.name}")
master-node@master-node:~$ echo https://127.0.0.1:8443/
https://127.0.0.1:8443/
master-node@master-node:~$ kubectl -n default port-forward $POD_NAME 8443:8443
Forwarding from 127.0.0.1:8443 -> 8443
Forwarding from :::8443 -> 8443
Handling connection for 8443
Handling connection for 8443
E0620 12:12:14.469035 11318 portforward.go:391] error copying from local connection to remote stream: read tcp4 127.0.0.1:8443->127.0.0.1:59070: read: connection reset by peer
Handling connection for 8443
E0620 12:16:21.813494 11318 portforward.go:378] error copying from remote stream to local connection: readfrom tcp4 127.0.0.1:8443->127.0.0.1:59120: write tcp4 127.0.0.1:8443->127.0.0.1:59120: write: broken pipe

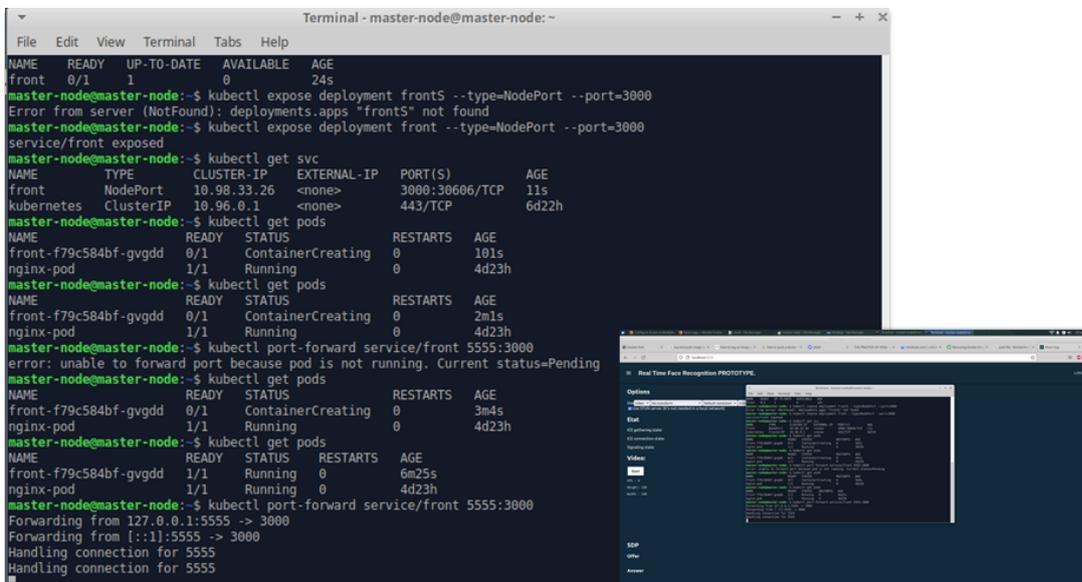
```

Figure 4.27: Installer Helm

Déployer le frontend



```
1  apiVersion: v1
2  kind: ServiceAccount
3  metadata:
4    name: admin-user
5    namespace: kubernetes-dashboard
6
7  ---
8  apiVersion: rbac.authorization.k8s.io/v1
9  kind: ClusterRoleBinding
10 metadata:
11   name: admin-user
12 roleRef:
13   apiGroup: rbac.authorization.k8s.io
14   kind: ClusterRole
15   name: cluster-admin
16 subjects:
17 - kind: ServiceAccount
18   name: admin-user
19   namespace: kubernetes-dashboard
```



```
Terminal - master-node@master-node: ~
NAME READY UP-TO-DATE AVAILABLE AGE
front 0/1 1 0 24s
master-node@master-node:~$ kubectl expose deployment fronts --type=NodePort --port=3000
Error from server (NotFound): deployments.apps "fronts" not found
master-node@master-node:~$ kubectl expose deployment front --type=NodePort --port=3000
service/front exposed
master-node@master-node:~$ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
front NodePort 10.98.33.26 <none> 3000:30606/TCP 11s
kubernetes ClusterIP 10.98.0.1 <none> 443/TCP 6d22h
master-node@master-node:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
front-f79c584bf-gvgdd 0/1 ContainerCreating 0 101s
nginx-pod 1/1 Running 0 4d23h
master-node@master-node:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
front-f79c584bf-gvgdd 0/1 ContainerCreating 0 2m1s
nginx-pod 1/1 Running 0 4d23h
master-node@master-node:~$ kubectl port-forward service/front 5555:3000
error: unable to forward port because pod is not running. Current status=Pending
master-node@master-node:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
front-f79c584bf-gvgdd 0/1 ContainerCreating 0 3m4s
nginx-pod 1/1 Running 0 4d23h
master-node@master-node:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
front-f79c584bf-gvgdd 1/1 Running 0 6m25s
nginx-pod 1/1 Running 0 4d23h
master-node@master-node:~$ kubectl port-forward service/front 5555:3000
Forwarding from 127.0.0.1:5555 -> 3000
Forwarding from [::]:5555 -> 3000
Handling connection for 5555
Handling connection for 5555
```

Figure 4.29: Déployer le dashboard kubernetes

Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
front-7778f665-dccq8	po001ent/front	app: front pod-template-hash: 7778f665	worker2-node	Running	0	-	-	58 seconds ago
dashboard-metrics-scraper-8c474659-cr9p	kubernetes/metrics-scraper-v1.0.0	k8s-app: dashboard-metrics-scraper pod-template-hash: 8c47465d	worker2-node	Running	0	-	-	an hour ago
kubernetes-dashboard-5676d8b85-95vd	kubernetes/dashboard-v2.6.0	k8s-app: kubernetes-dashboard pod-template-hash: 5676d8b85	worker2-node	Running	0	-	-	an hour ago
kubernetes-dashboard-7f6c9b264-87gp	kubernetes/dashboard-v2.6.0	app: kubernetes.io/component.kubernetes.io/instance: kubernetes.io/managed-by: kubernetes.io/managed-by: front	worker2-node	Running	0	-	-	2 hours ago
nginx-pod	nginx	-	worker2-node	Running	0	-	-	5 days ago
calico-kube-controllers-544f5556-9apd9	docker.io/calico/kube-controllers-v3.23.0	k8s-app: calico-kube-controllers pod-template-hash: 544f555d	worker2-node	Running	0	-	-	7 days ago
calico-node-h78bb	docker.io/calico/node:v3.23.0	controller-revision-hash: 5696b4769 k8s-app: calico-node pod-template-generation: 1	worker2-node	Running	1	-	-	7 days ago
kube-proxy-25kt	k8s.gcr.io/kube-proxy-v1.24.1	controller-revision-hash: 58f5dfb47 k8s-app: kube-proxy pod-template-generation: 1	worker2-node	Running	1	-	-	7 days ago

Figure 4.30: Dashboard kubernetes

4.4 Conclusion

Dans ce chapitre, nous avons pu implémenter les trois microservices qui composent notre système : Frontend, coturn et BackendML, en utilisant minikube, kubeadm et kubeflow qui proposent des services différents afin de réaliser un déploiement local.

Chapter 5

Déploiement cloud

5.4 Conclusion

Dans ce chapitre, nous avons montré comment utiliser et configurer des infrastructures cloud, notamment AWS et Azure, pour créer des clusters, des rôles, des groupes de ressources, connecter des clusters à une machine et déployer différents systèmes (images simples, applications, modèles machine learning).

5.1 Introduction

Dans cette section, nous allons démontrer les différentes étapes de déploiement cloud que nous avons prises sur deux fournisseurs de services cloud : Amazon web services et Microsoft Azure.

5.2 Avec amazon web services (AWS)

AWS est le principal fournisseur de cloud sur le marché, offrant plus de 200 services complets et fournissant une large gamme d'opérations à la demande telles que la puissance de calcul, le stockage de base de données, la diffusion de contenu, etc. Pour aider les entreprises à se développer et à prospérer.[32]

Aws nous propose deux façons de démarrer, d'exécuter et de mettre à l'échelle Kubernetes:

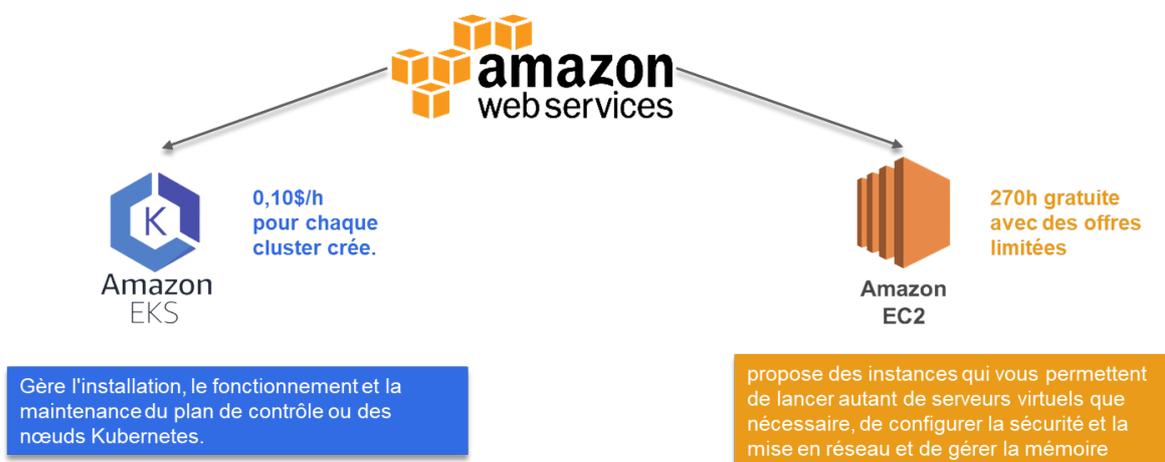


Figure 5.1: AWS Kubernetes

1. **EKS:** vous permet d'exécuter Kubernetes sans avoir à installer, utiliser et maintenir votre propre plan de contrôle ou nœuds Kubernetes.

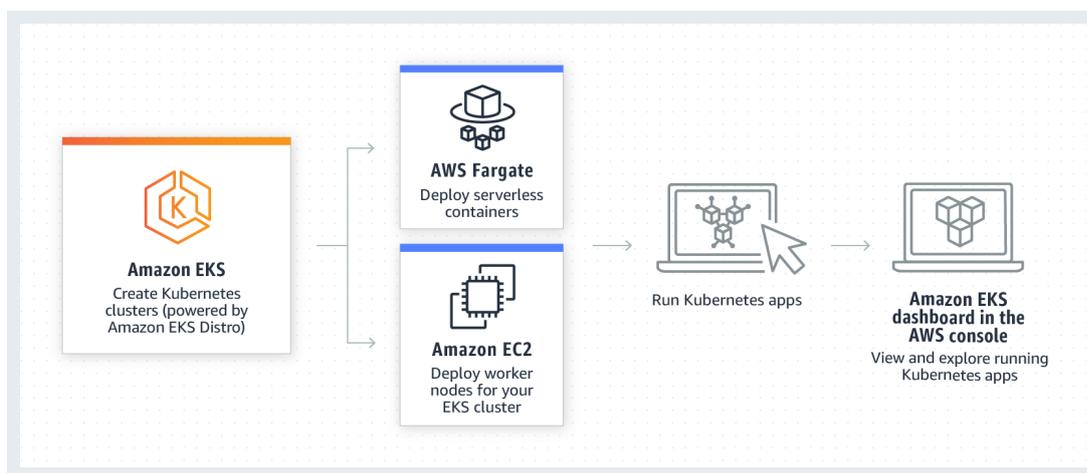


Figure 5.2: Amazon Eks

- **Creation d'un cluster Eks:** Pour créer un cluster Eks, vous devez d'abord créer un VPC (Amazon Virtual Private Cloud) qui vous permet de lancer des ressources AWS dans un réseau virtuel que vous définissez. Ensuite vous crée un rôle IAM qui sera utilisé pour accéder au cluster.

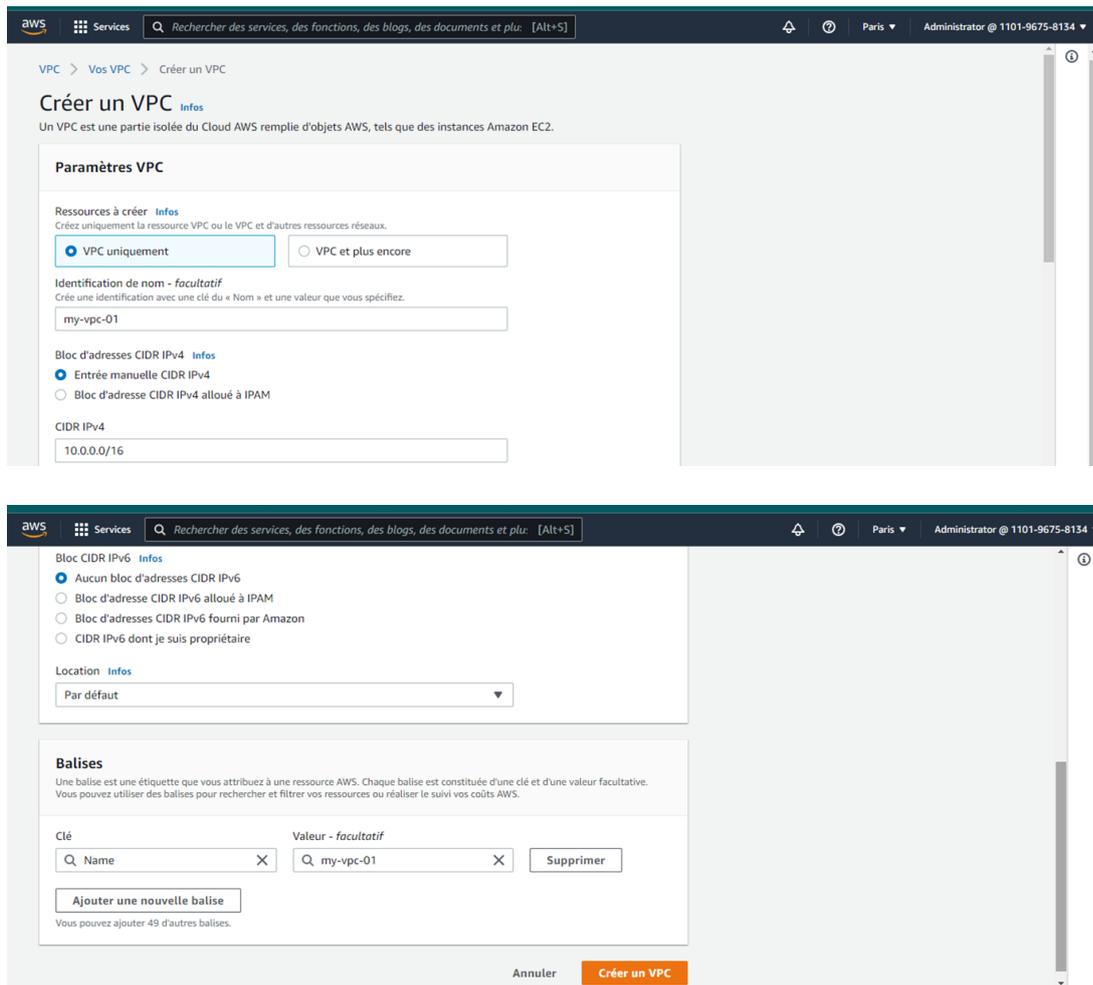


Figure 5.3: Crée un vpc

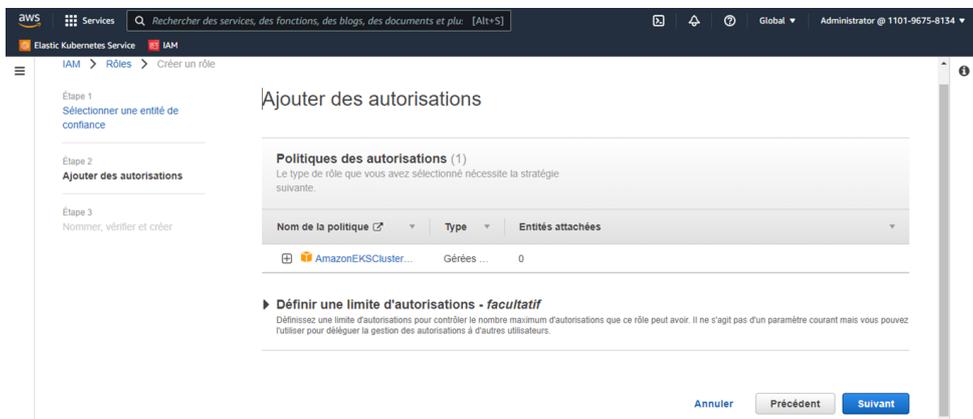
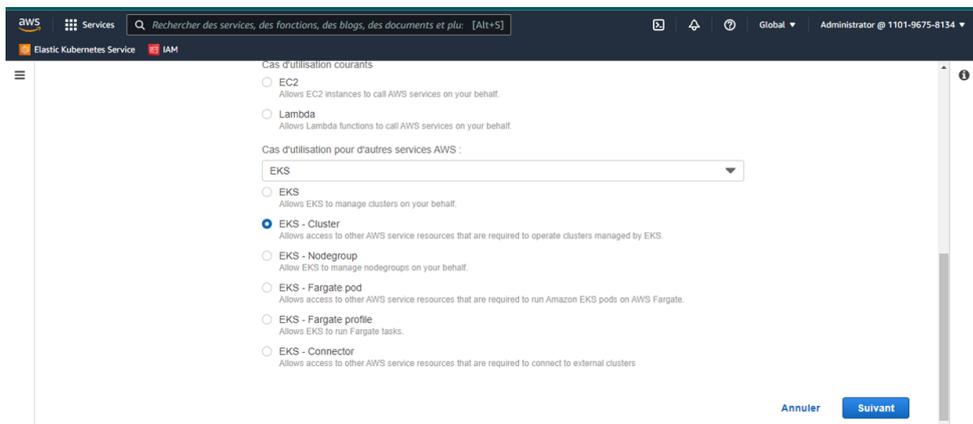
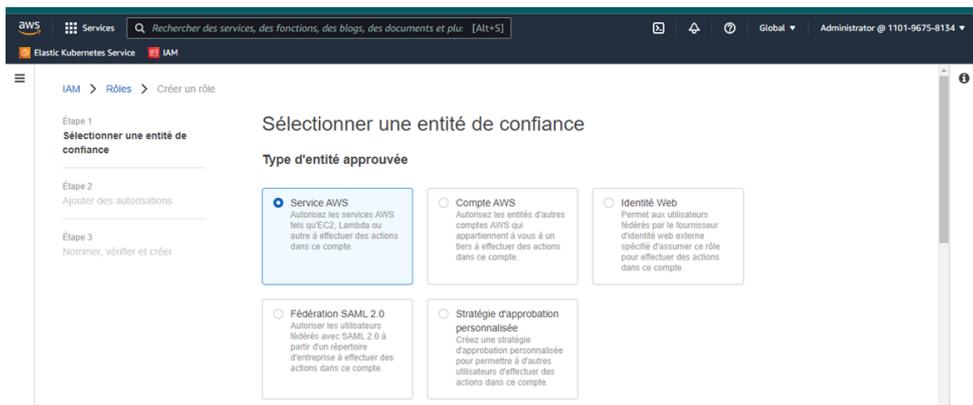
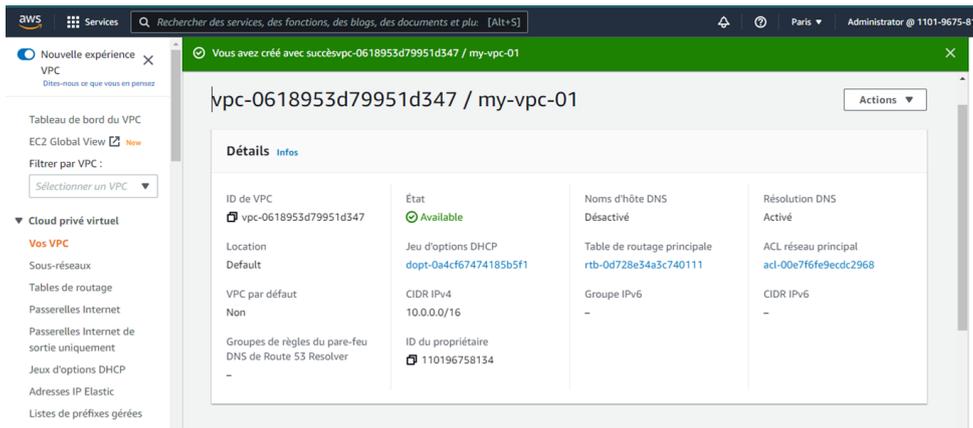


Figure 5.4: Crée un vpc

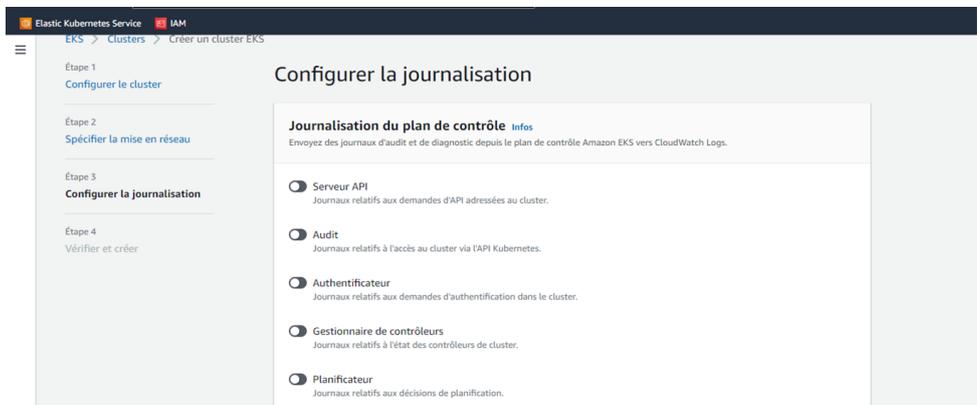
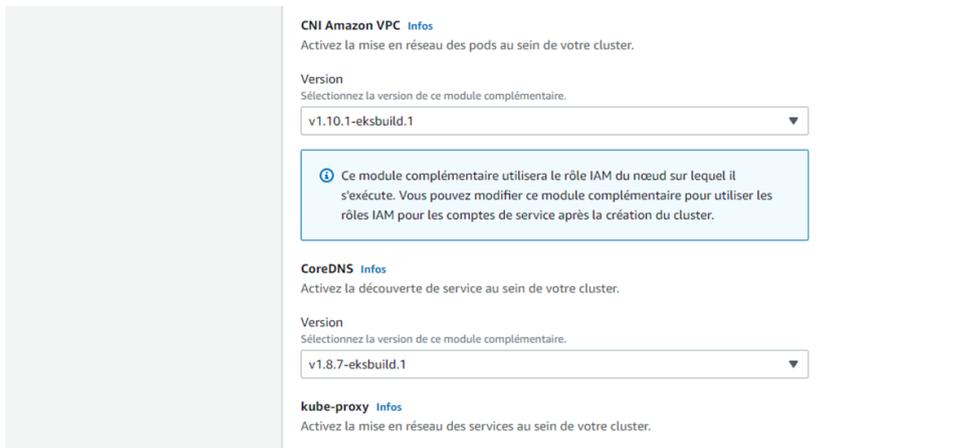
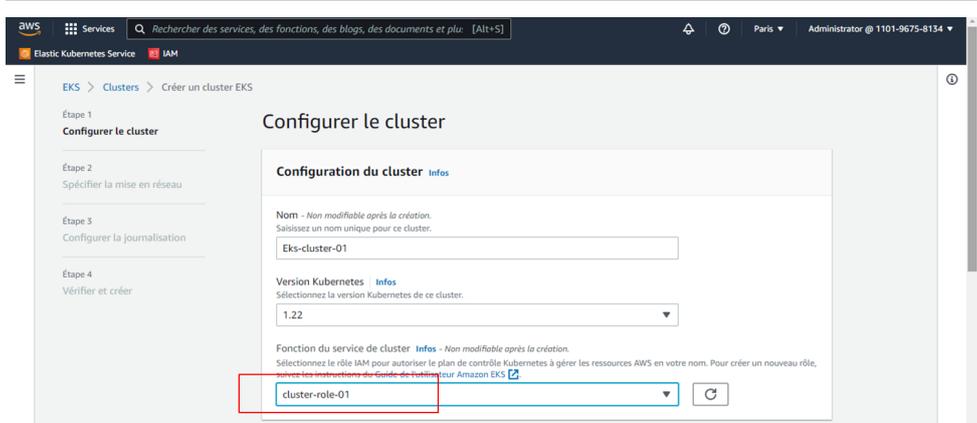
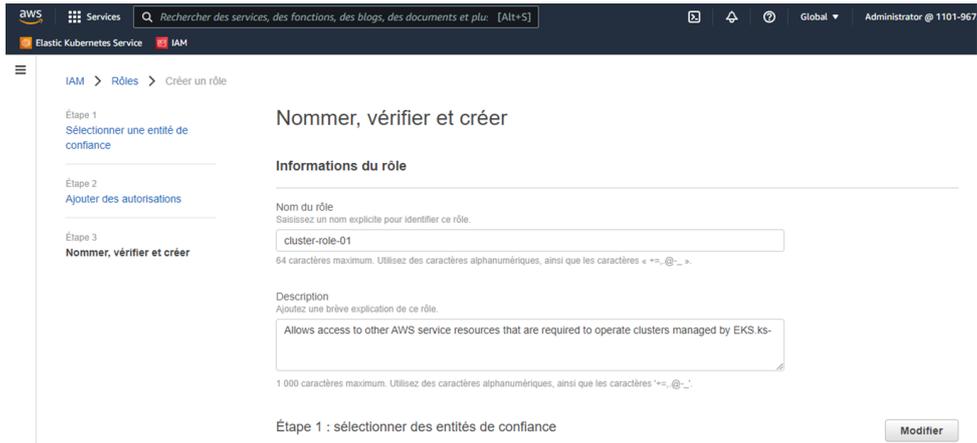


Figure 5.5: Créer un cluster

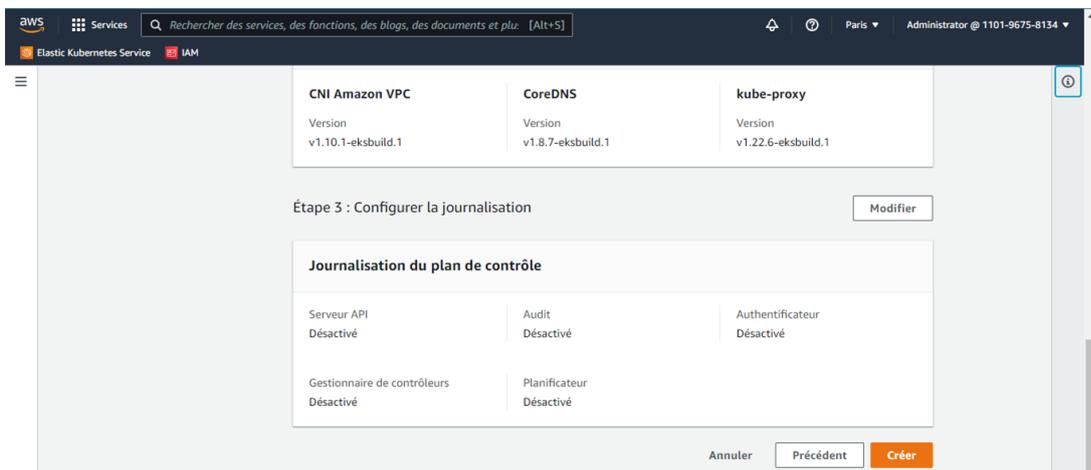
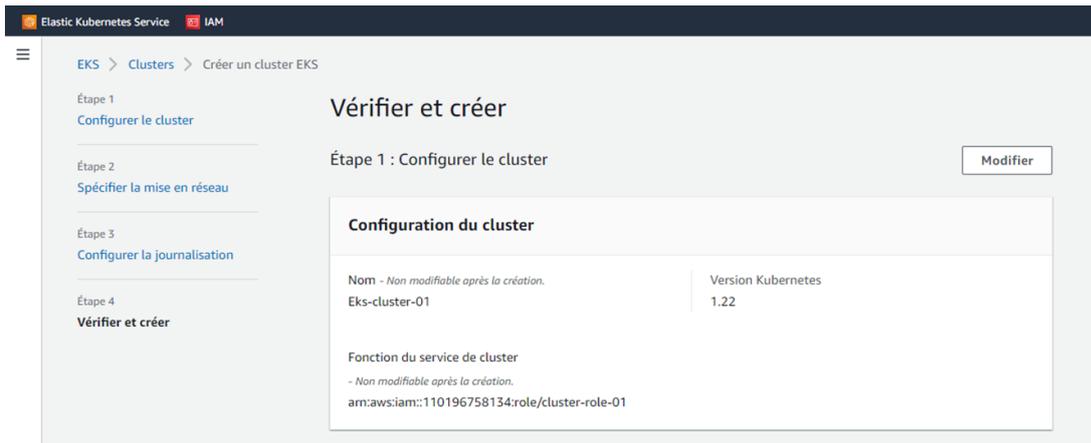


Figure 5.6: Créer un cluster

2. **EC2:** Vous permet de Gérer entièrement votre déploiement.on fournissant des instances de calcul qui hébergent vos nœuds Kubernetes.

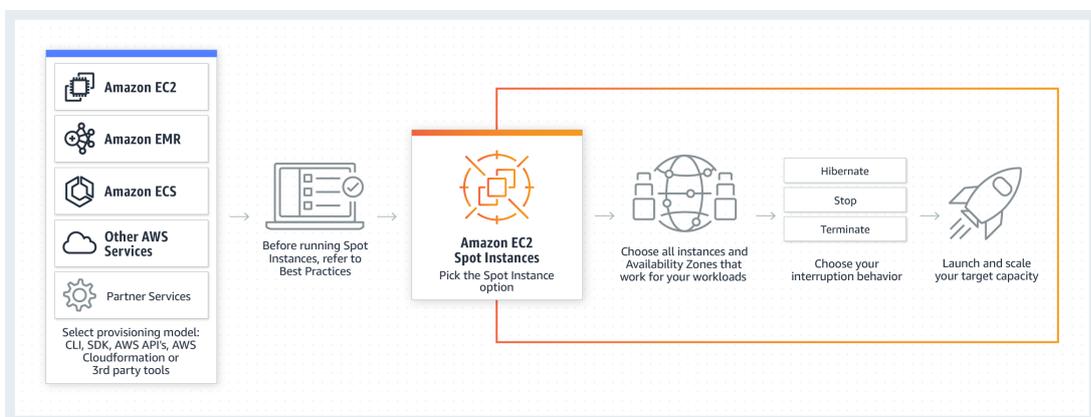


Figure 5.7: Amazon EC2

- Création d'un utilisateur IAM:** Afin de réaliser toute action sur AWS, il est obligé de s'identifier auprès d'IAM qui est un service de gestion des identités et des accès, il gère les autorisations et les interdictions des actions selon les droits accordés par l'administrateur du compte.

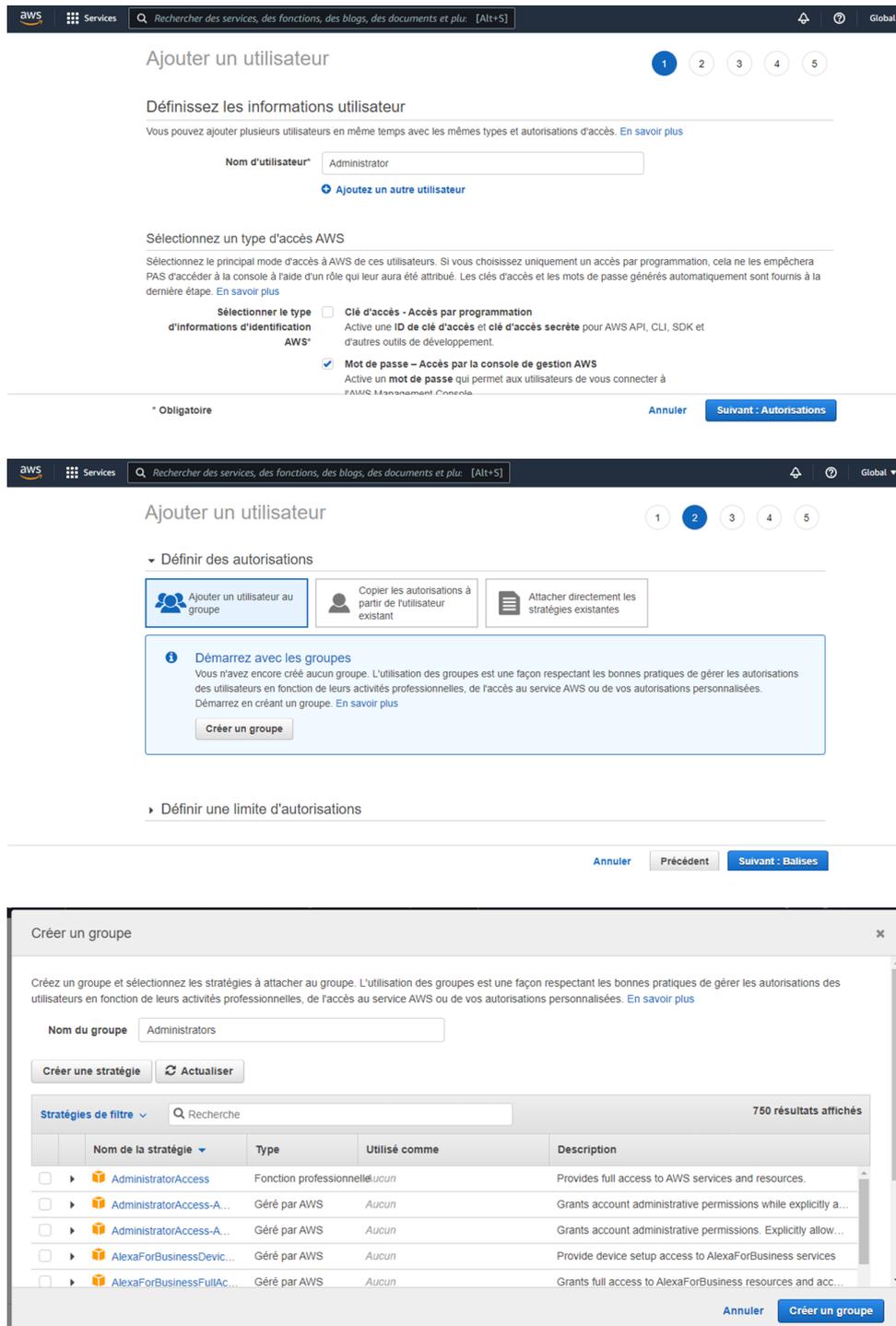


Figure 5.8: IAM

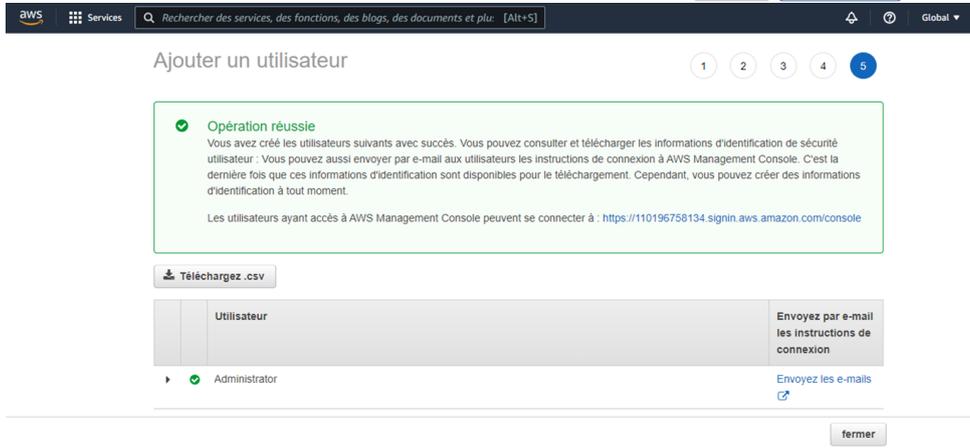
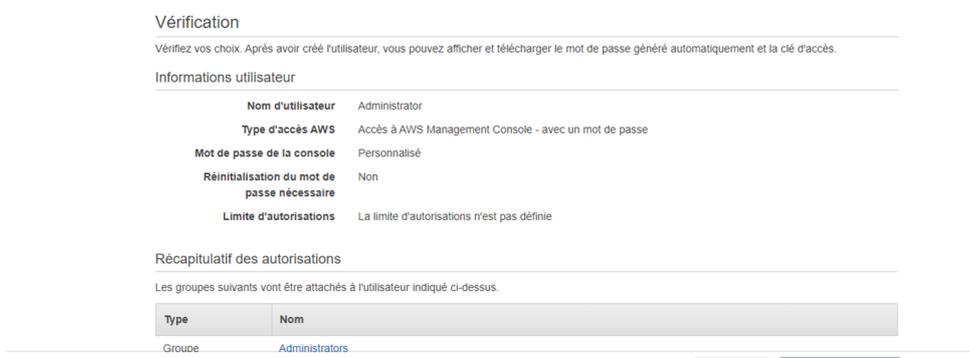
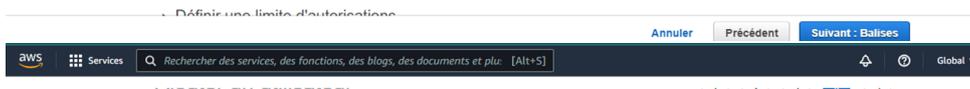
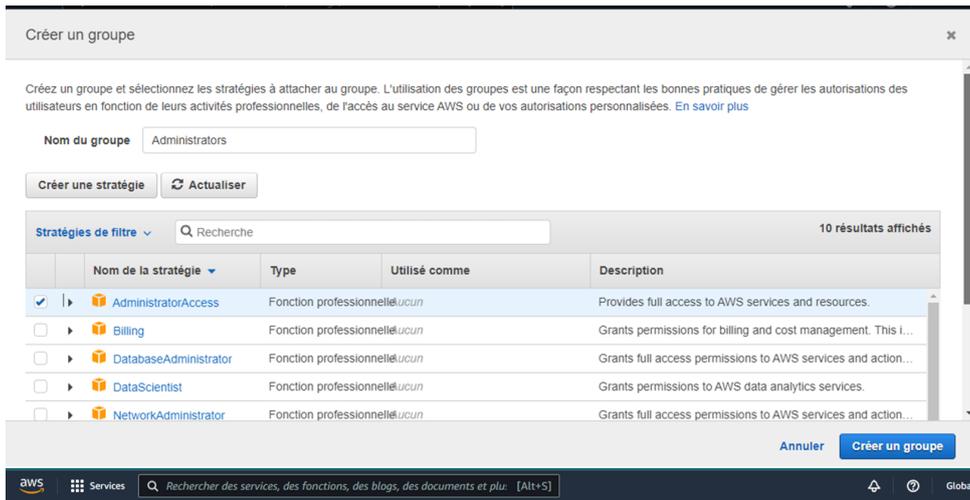


Figure 5.9: IAM

- **Création d'instances EC2:** Une instance EC2 est un serveur virtuel hébergé dans Elastic Compute Cloud (EC2) pour exécuter des applications sur AWS.

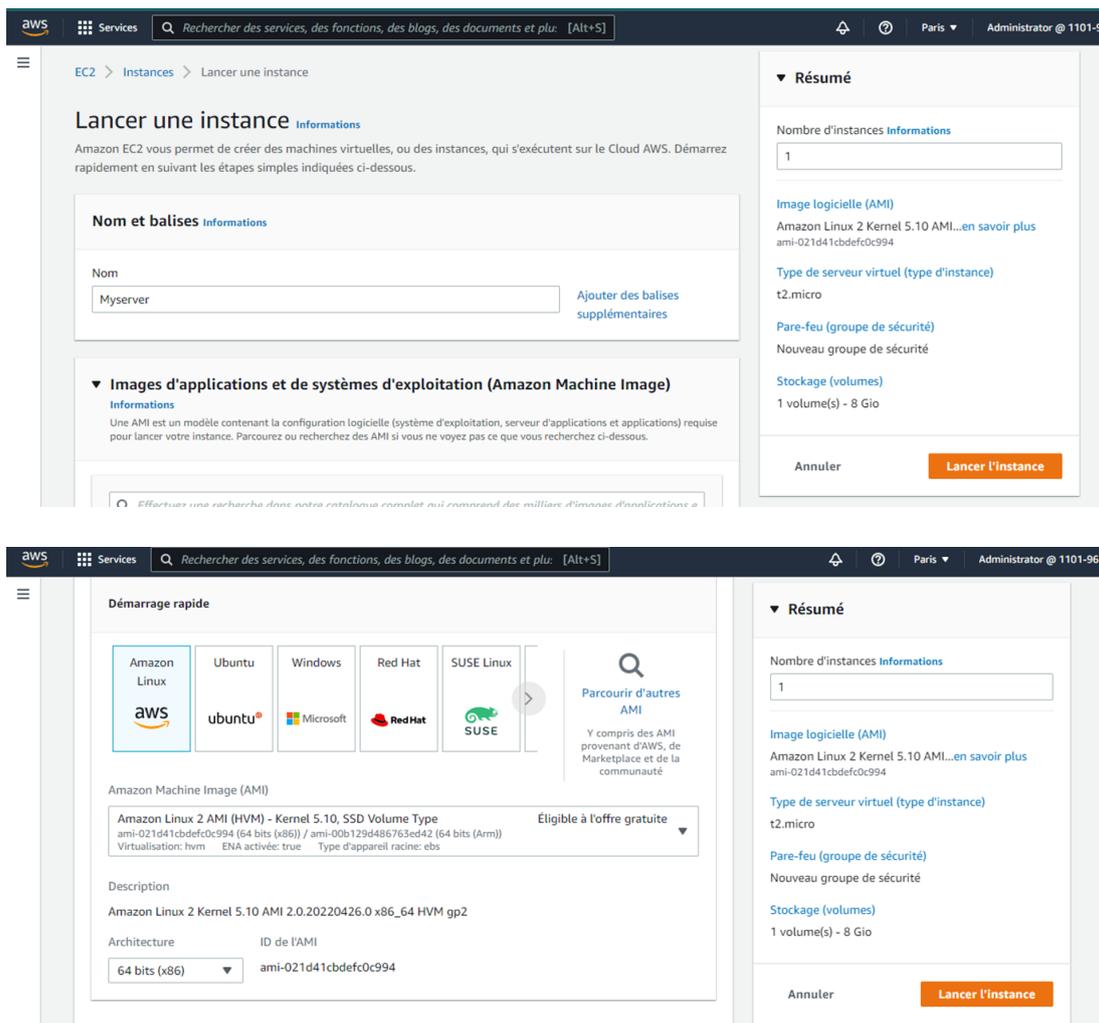


Figure 5.10: Instance EC2

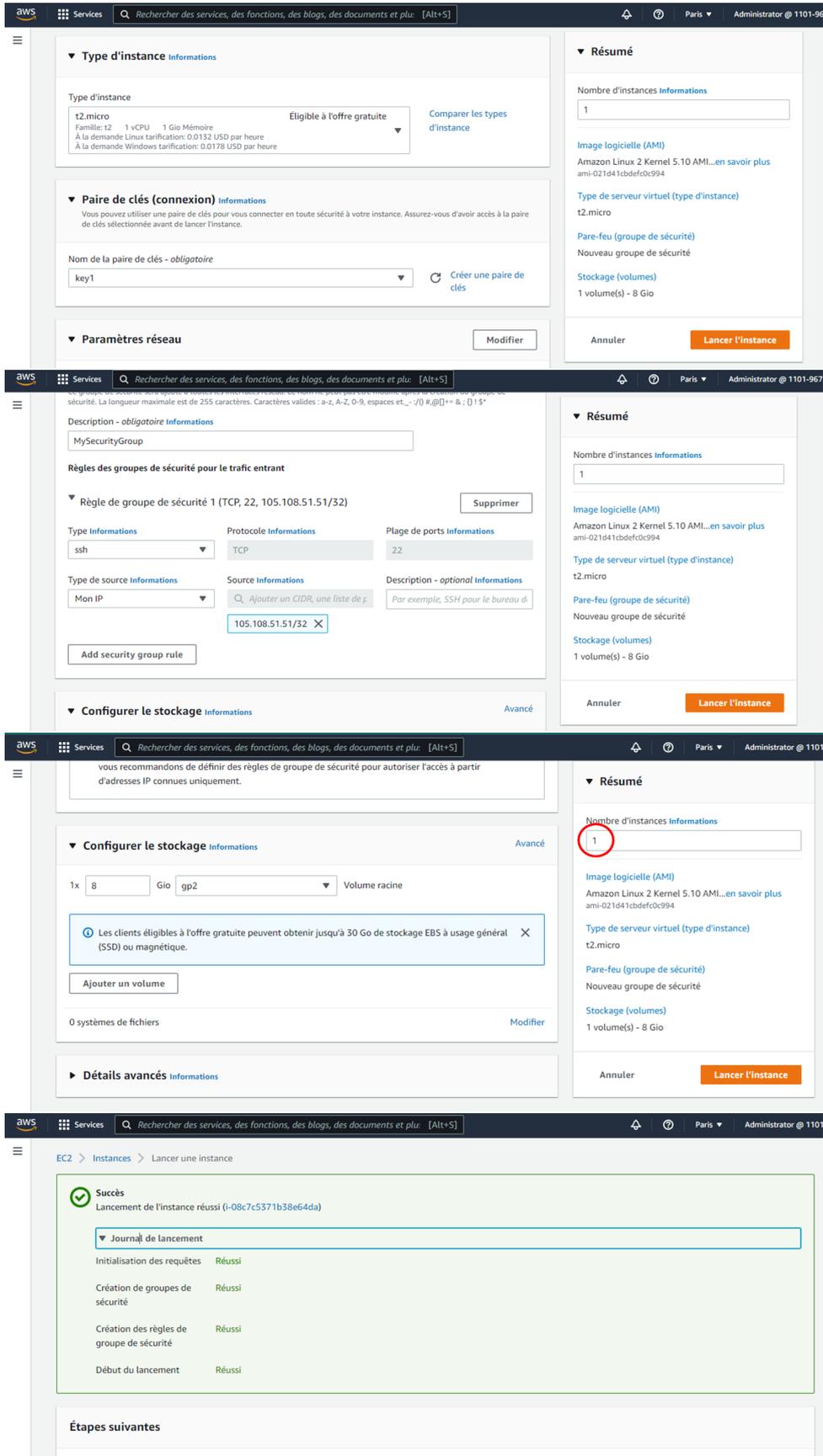


Figure 5.11: Instance EC2

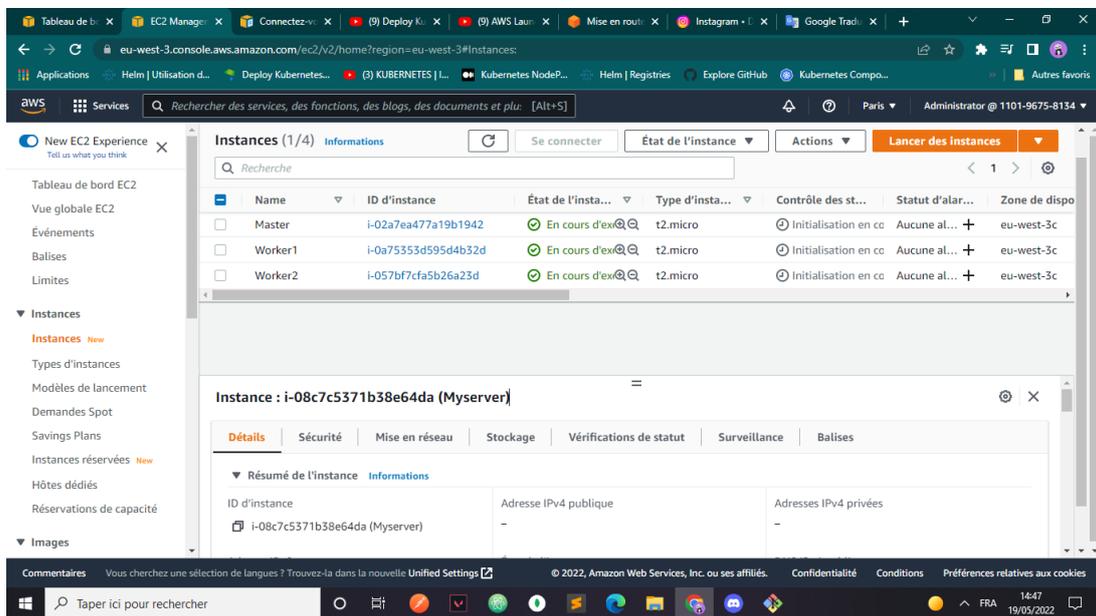


Figure 5.12: Les trois instances créées

- **Installation:** Nous procéderons à l'installation des outils nécessaires sur les trois instances créées.

Installation de docker

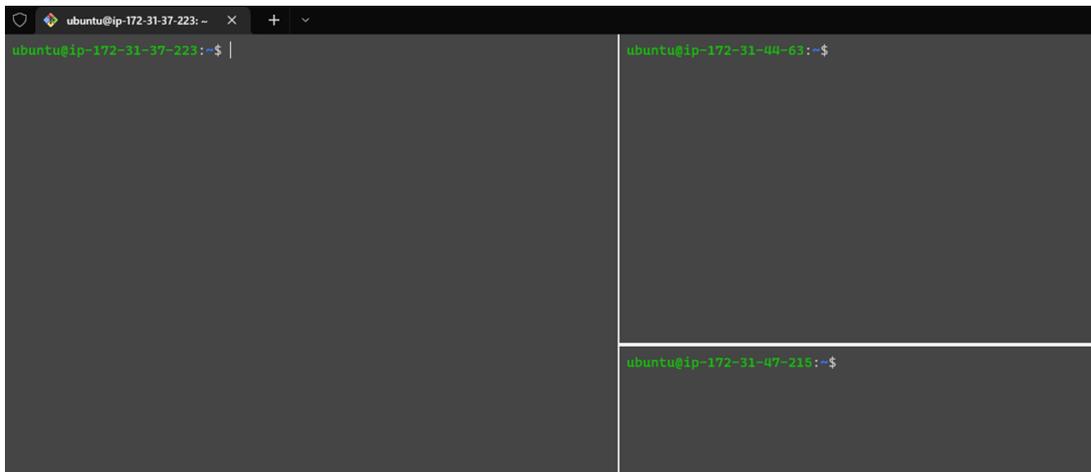
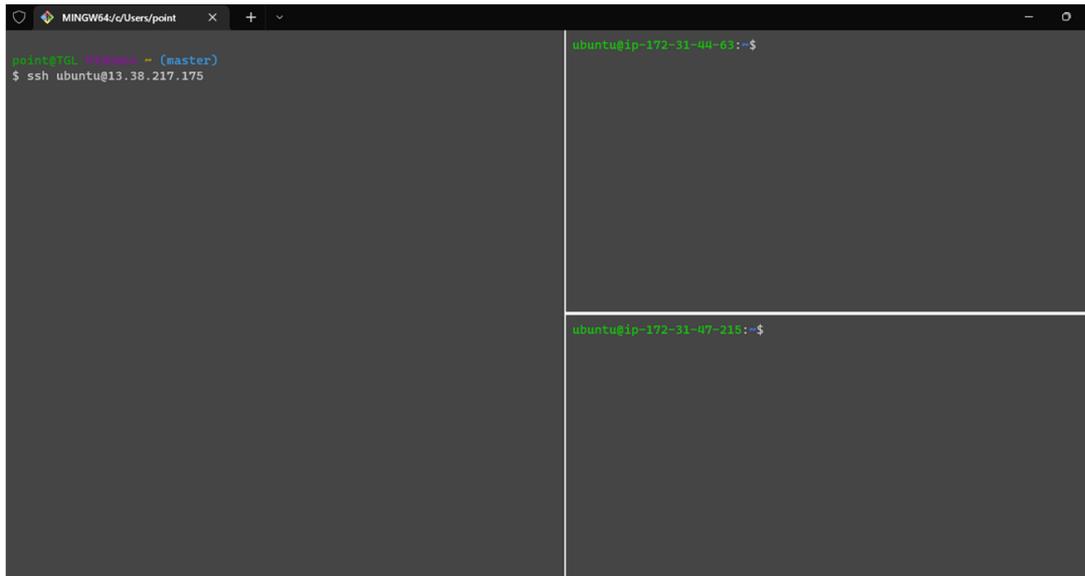


Figure 5.13: Installation sur les trois instances

```

ubuntu@ip-172-31-37-223:~$ sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:2 http://eu-west-3.ec2.archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:3 http://eu-west-3.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [109 kB]
Get:4 http://eu-west-3.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [99.8 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [126 kB]
Get:6 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [33.0 kB]
Get:7 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [1956 B]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 Packages [124 kB]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/restricted Translation-en [18.7 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 c-n-f Metadata [456 B]
Get:11 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [59.5 kB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/universe Translation-en [18.7 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [960 B]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 Packages [4192 B]
Get:15 http://security.ubuntu.com/ubuntu jammy-security/multiverse Translation-en [900 B]
Get:16 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64

ubuntu@ip-172-31-44-63:~$ sudo apt-get update

ubuntu@ip-172-31-47-213:~$ sudo apt-get update

ubuntu@ip-172-31-37-223:~$ sudo mkdir -p /etc/apt/keyrings
ubuntu@ip-172-31-37-223:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
ubuntu@ip-172-31-37-223:~$

ubuntu@ip-172-31-44-63:~$ sudo mkdir -p /etc/apt/keyrings
ubuntu@ip-172-31-44-63:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
ubuntu@ip-172-31-44-63:~$

ubuntu@ip-172-31-47-213:~$ sudo mkdir -p /etc/apt/keyrings
ubuntu@ip-172-31-47-213:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
ubuntu@ip-172-31-47-213:~$

ubuntu@ip-172-31-37-223:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras docker-scan-plugin libltdl7 libsllrp0 pigz sllrp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin docker-scan-plugin libltdl7 libsllrp0 pigz sllrp4netns
0 upgraded, 10 newly installed, 0 to remove and 42 not upgraded.
Need to get 108 MB of archives.
After this operation, 449 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 https://download.docker.com/linux/ubuntu jammy/stable amd64 containerd.io amd64 1.6.4-1 [28.1 MB]
Get:2 http://eu-west-3.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6-1 [63.6 kB]
Get:3 http://eu-west-3.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libltdl7 amd64 2.4.6-15build2 [39.6 kB]
Get:4 http://eu-west-3.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 libsllrp0 amd64 4.6.1-1build1 [61.5 kB]
Get:5 http://eu-west-3.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 sllrp4netns amd64 1.0.1-2 [28.2 kB]
Get:6 https://download.docker.com/linux/ubuntu jammy/stable amd64 docker-ce-cli amd64 5:20.10.16~3~0-ubuntu-jammy [40.6 MB]
Get:7 https://download.docker.com/linux/ubuntu jammy/stable amd64 docker-ce amd64 5:20.10.16~3~0-ubuntu-jammy [21.0 MB]
63% [7 docker-ce 0 B/21.0 MB 0%]

containerd.io docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-ce-rootless-extras docker-scan-plugin libltdl7 libsllrp0 pigz sllrp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin docker-scan-plugin libltdl7 libsllrp0 pigz sllrp4netns
0 upgraded, 10 newly installed, 0 to remove and 42 not upgraded.
Need to get 108 MB of archives.
After this operation, 449 MB of additional disk space will be used.
Do you want to continue? [Y/n]
ubuntu@ip-172-31-47-213:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin

```

Figure 5.14: Installation sur les trois instances


```

ubuntu@ip-172-31-47-215:~$ kubectl version
No containers need to be restarted.
No user sessions are running outdated binaries.
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-44-63:~$ sudo apt-mark hold kubeadm kubelet kubectl
kubeadm set on hold.
kubelet set on hold.
kubectl set on hold.
ubuntu@ip-172-31-44-63:~$ kubectl version
Kubernetes v1.24.1
ubuntu@ip-172-31-47-215:~$ sudo apt-get install kubeadm kubelet kubectl

```

Initialisation de kubeadm

```

ubuntu@master-node:~$ sudo kubeadm init
[init] Using Kubernetes version: v1.24.1
[preflight] Running pre-flight checks
[WARNING SystemVerification]: missing optional cgroups: blkio
error execution phase preflight: [preflight] Some fatal errors occurred:
[ERROR NumCPU]: the number of available CPUs 1 is less than the required 2
[ERROR Mem]: the system RAM (967 MB) is less than the minimum 1700 MB
[ERROR CRI]: container runtime is not running: output: E0527 14:41:11.797952 14185 remote_runtime.go:925] "Status from runtime service failed" err="rpc error: code = Unimplemented desc = unknown service runtime.v1alpha2.RuntimeService" time="2022-05-27T14:41:11Z" level=fatal msg="getting status of runtime: rpc error: code = Unimplemented desc = unknown service runtime.v1alpha2.RuntimeService" , error: exit status 1
[preflight] If you know what you are doing, you can make a check non-fatal with '--ignore-preflight-errors=...'
To see the stack trace of this error execute with --v=5 or higher
ubuntu@master-node:~$

```

Erreur

Figure 5.16: Installation sur les trois instances

Type d'inst...	vCPU	Architecture	Mémoire (GiB)	Stockage (Go)	Type de stockage	Performances réseau	À la demande Linux tarifation
t2.nano	1	i386, x86_64	0.5	-	-	Low to Moderate	0.0066 USD par heure
t2.micro	1	i386, x86_64	1	-	-	Low to Moderate	0.0132 USD par heure
t2.small	1	i386, x86_64	2	-	-	Low to Moderate	0.0264 USD par heure
t2.medium	2	i386, x86_64	4	-	-	Low to Moderate	0.0528 USD par heure
t2.large	2	x86_64	8	-	-	Low to Moderate	0.1056 USD par heure
t2.xlarge	4	x86_64	16	-	-	Moderate	0.2112 USD par heure
t2.2xlarge	8	x86_64	32	-	-	Moderate	0.4224 USD par heure
t3.nano	2	x86_64	0.5	-	-	Up to 5 Gigabit	0.0059 USD par heure

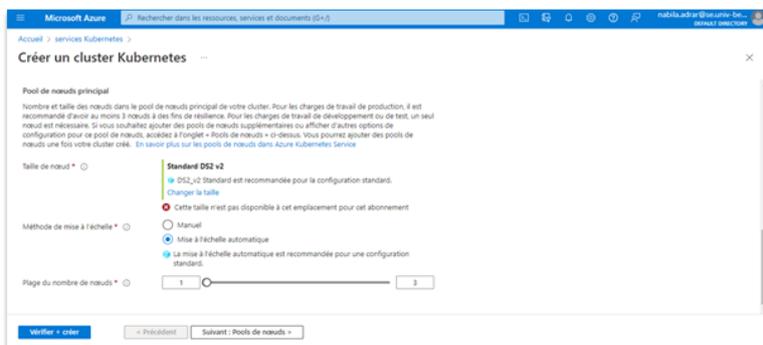
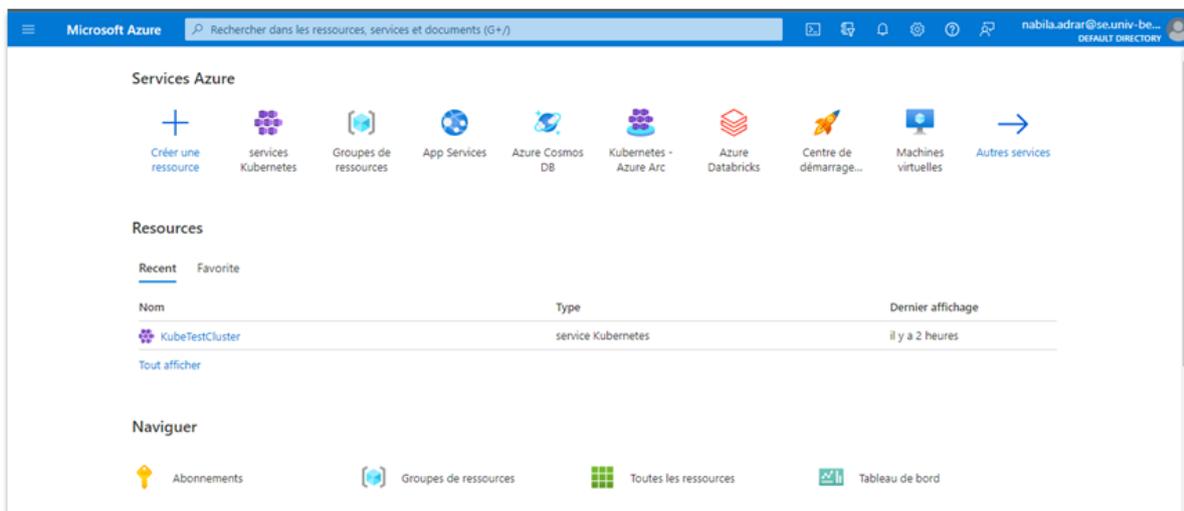
RQ : le type d'instance inclus dans l'offre gratuite est t2.micro qui possède une capacité limitée qui ne répondait pas à nos besoins

Figure 5.17: Types d'instances

5.3 Avec microsoft azure

Azure est la plate-forme cloud de Microsoft, rassemblant plus de 200 services cloud différents, notamment le calcul, l'analyse, le stockage et la mise en réseau. Azure Kubernetes Service (AKS) offre le moyen le plus rapide de commencer à développer et à déployer des applications cloud natives.

Se connecter au compte azure



← Crée un cluster Avec l'interface azure

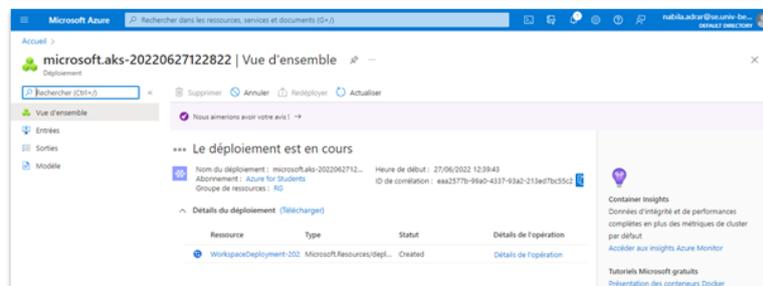


Figure 5.18: Crée un cluster avec l'interface Azure

Crée un cluster avec Azure CLI

```
az login
az account set --subscription b0dc7a2e-a794-4dce-9083-d3deb5796c7
```

```
az group create -n MyRGoupe -l uksouth
```

```
az aks create -g MyRGoupe -n KubeTestCluster -s Standard_D4s_v3 -c
3 -l uksouth --generate-ssh-keys
```

```
MINGW64/
MINGW64/c/Users/point
$ az login
* [93mA web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in the web browser. If no web browser is available or if the web browser fails to open, use device code flow with 'az login --use-device-code'.*]
[
  {
    "cloudName": "AzureCloud",
    "homeTenantId": "768fe75e-22f2-45f0-b34d-2c6a98c6f1dd",
    "id": "b0dc7a2e-a794-4dce-9083-d3deb5796c7b",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Azure for Students",
    "state": "Enabled",
    "tenantId": "768fe75e-22f2-45f0-b34d-2c6a98c6f1dd",
    "user": {
      "name": "nabila.adrar@se.univ-bejaia.dz",
      "type": "user"
    }
  }
]
MINGW64/
MINGW64/ (master)
$ az group create -n MyRGoupe -l UKSouth
{
  "id": "/subscriptions/b0dc7a2e-a794-4dce-9083-d3deb5796c7b/resourceGroups/MyRGoupe",
  "location": "uksouth",
  "managedBy": null,
  "name": "MyRGoupe",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

```
MINGW64/
MINGW64/c/Users/point
MINGW64/ (master)
$ az aks create -g MyRGoupe -n KubeTestCluster -s Standard_D4s_v3 -c 1 -l UKSouth --generate-ssh-keys
[K] Finished ..
{
  "aadProfile": null,
  "addonProfiles": null,
  "agentPoolProfiles": [
    {
      "availabilityZones": null,
      "count": 1,
      "creationData": null,
      "currentOrchestratorVersion": "1.22.6",
      "enableAutoScaling": false,
      "enableEncryptionAtHost": false,
      "enableFips": false,
      "enableNodePublicIp": false,
      "enableUltraSsd": false,
      "gpuInstanceProfile": null,
      "kubenetConfig": null,
      "kubenetDiskType": "OS",
      "linuxOsConfig": null,
      "maxCount": null,
      "maxPods": 110,
      "minCount": null,
      "mode": "System",
      "name": "nodepool1",
      "nodeImageVersion": "AKSUBuntu-1804gen2containerd-2022.06.13",
      "nodeLabels": null,
      "nodePublicIpPrefixId": null,
      "nodeTaints": null,
      "orchestratorVersion": "1.22.6",
      "osDiskSizeGb": 128,
      "osDiskType": "Managed",
      "osSku": "Ubuntu",
      "osType": "Linux",
    }
  ]
}
```

Figure 5.19: Crée un cluster avec Azure CLI

```

"userAssignedIdentities": null
},
"identityProfile": {
  "kubernetesIdentity": {
    "clientId": "9d2a13ea-cb8d-4592-8538-ef81be96fdd4",
    "objectId": "6de895a8-ba87-4aa9-b7a6-ff6d5e4b1c3",
    "resourceId": "/subscriptions/b8dc7a2e-a794-4dce-9083-d3deb5796c7b/resourcegroups/MC_MyRGroupe_KubeTestCluster_uksouth/providers/Microsoft.ManagedIdentity/userAssignedIdentities/KubeTestCluster-agentpool"
  }
},
"kubernetesVersion": "1.22.6",
"linuxProfile": {
  "adminUsername": "azureuser",
  "ssh": {
    "publicKeys": [
      {
        "keyData": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDAck6f1MmELgnMwIVp2nJEIknXeyzE1HlXnaUqJRCowXTx9o7H1fdkq5TORu3Gbc0ksh1kSLnsdmQZDHuo19PZHu0Wz8pVx0t8ZUMUip8n5Ulrk2MVTdyqHML28pof8Gz2Z8vYvOAGs+Gz/cZDSPhiUwZud8Rf3+1Bext53rpf1SLc8H+0BqJchyhu9p01Emm3jJwqBUK+6995SPpDj5bDAUK+M1Okqc019UeeW6Gceeg/0LUUp9d4b8dWk2AK7c1j5dD0/cjZwobaeR/ALDYz2dHAc65N0tZeh33qfF5j98V13qzIbuYQZc1c8mWpHz07R5v77m8v/4U/34wauLPz68r+rpR8Kw8f2jmmaPwIXQzhn2XjNSxpdg8Ppithgk4irV6C9VvSkDe8bCH4up/dqkFkH536Gw4SD1iikf+YH3EY1Npi+wrBA14ZqzCzWXMmdrsj6n9yH7dSB0Xf6G0vE4z8gMYshcDR5Rj8kcUk0wR5g1MoAs8= point@TGL\n"
      }
    ]
  }
},
"location": "uksouth",
"maxAgentPools": 100,
"name": "KubeTestCluster",
"networkProfile": {
  "dnsServiceIp": "10.0.0.10",
  "dockerBridgeCidr": "172.17.0.1/16",
  "ipFamilies": [
    "IPv4"
  ]
},
"loadBalancerProfile": {
  "allocatedOutboundPorts": null,

```

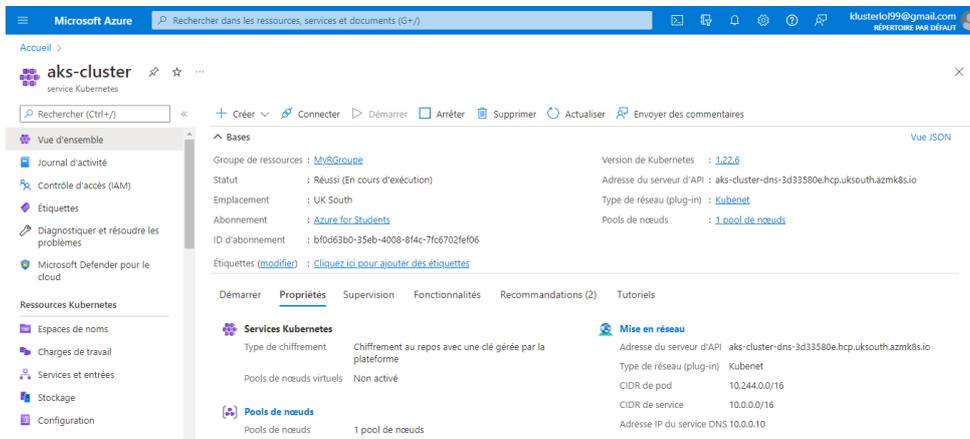


Figure 5.20: Le cluster créée

Connecter au cluster avec Azure CLI

```
az account set --subscription b0dc7a2e-a794-4dce-9083-d3deb5796c7b
```

```
az aks get-credentials --resource-group RG --name AKS-Cluster
```

```

PS C:\> az aks get-credentials --resource-group RG --name AKS-Cluster
Merged "AKS-Cluster" as current context in C:\Users\point\.kube\config
PS C:\> kubectl config get-contexts
CURRENT  NAME             CLUSTER          AUTHINFO          NAMESPACE
*        AKS-Cluster      AKS-Cluster      clusterUser_RG_AKS-Cluster
docker-desktop  docker-desktop  docker-desktop
PS C:\> kubectl config get-clusters
NAME
docker-desktop
AKS-Cluster
PS C:\> |

```

Figure 5.21: Se connecter au cluster

Pour commencer, nous allons tester les deux méthodes que nous propose azure: créer une application de base et créer une application d'image unique.

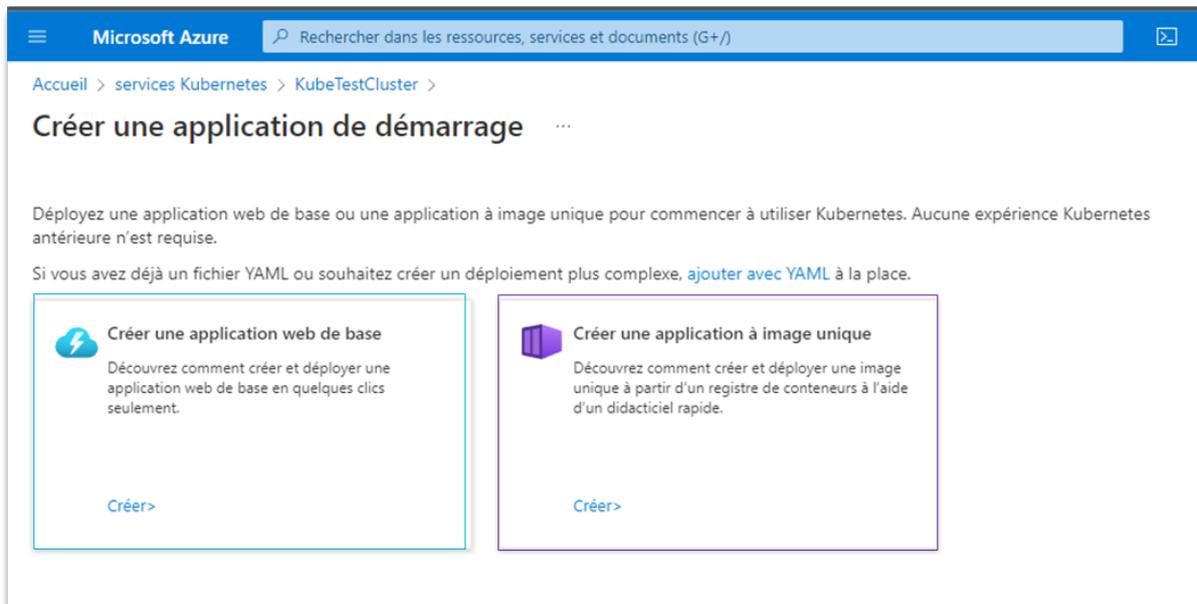


Figure 5.22: Deux méthodes de déploiement

1. **créer une application de base:** Pour bien comprendre le fonctionnement de cette étape, nous avons implémenté une image simple fournie par Azure, située dans `mcr.microsoft.com`, qui est le registre d'artefacts de Microsoft.

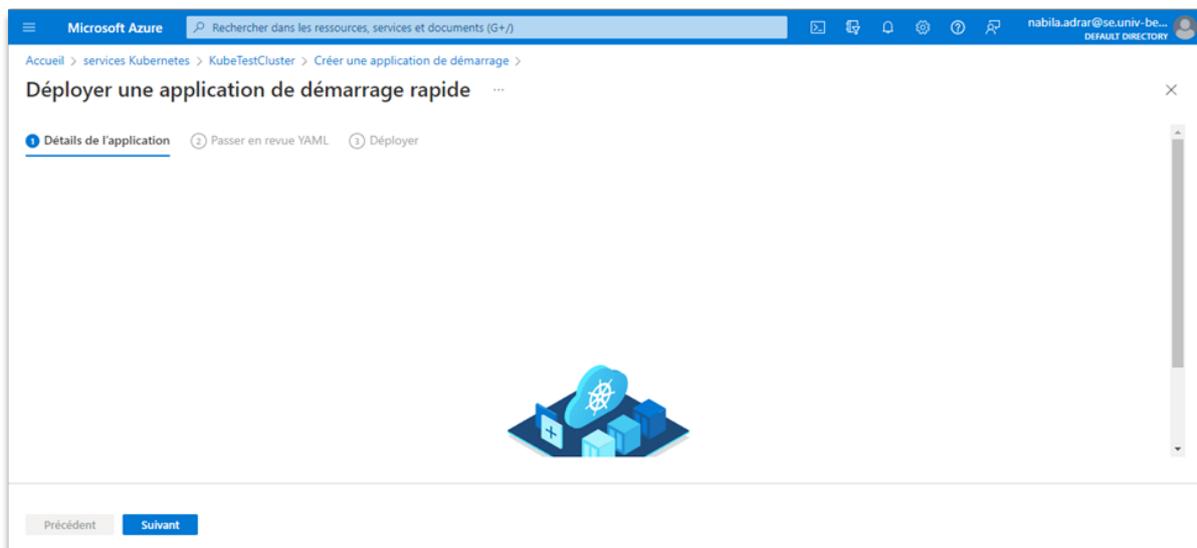


Figure 5.23: Méthode une

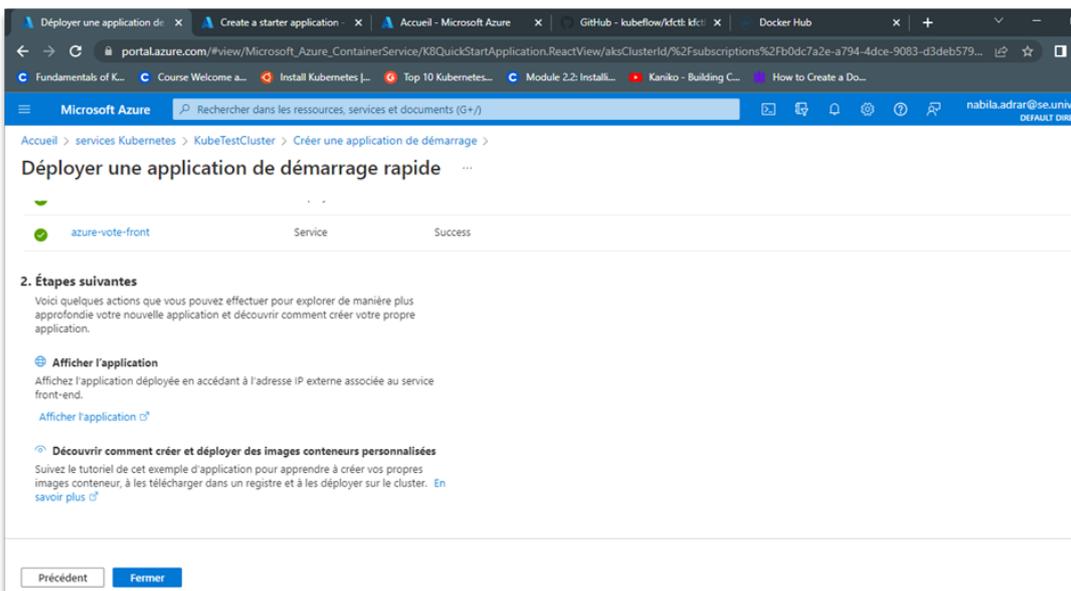
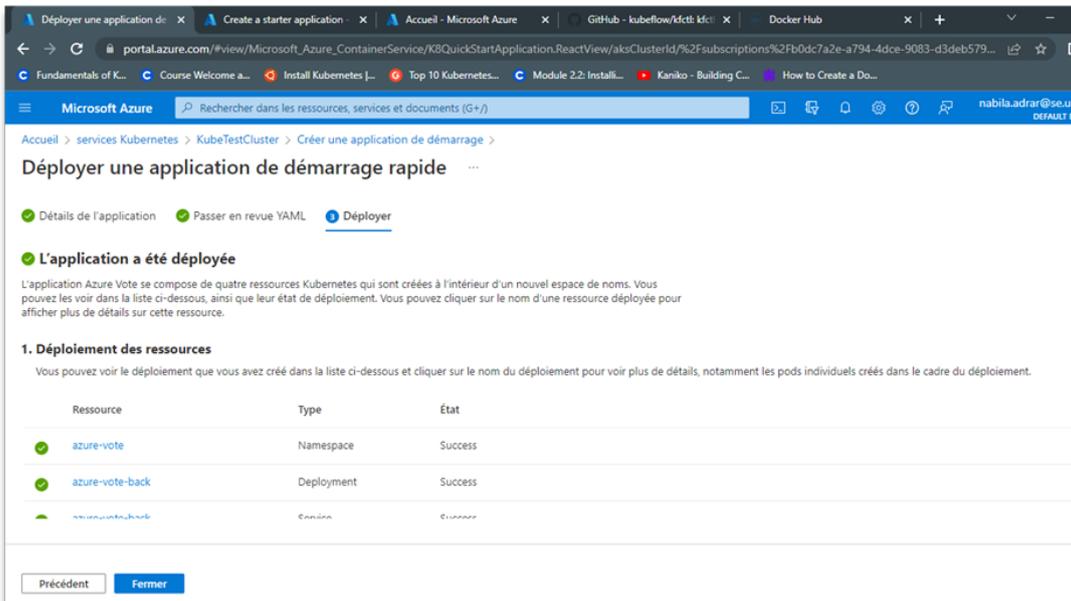
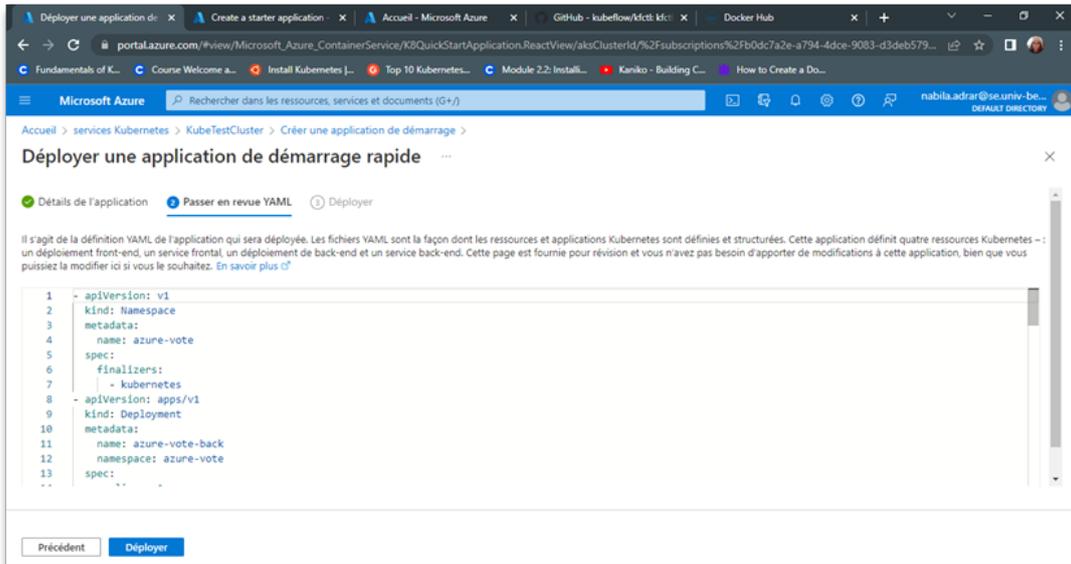


Figure 5.24: Méthode une étapes

2. créer une application d'image unique:

Ici, nous allons utiliser le registre de conteneurs Azure, pour cela Nous allons créer un registre Azure puis nous taguerons nos images locales puis nous pousserons les images vers le registre créé pour enfin les déployer.

The screenshot shows the Microsoft Azure portal interface. The main window displays the 'Créer une application à image unique' wizard, which is currently at the 'Obtenir l'image' step. The wizard includes sections for 'Détails du registre de conteneurs' (where 'Azure Container Registry' is selected) and 'Détails de l'image'. A 'Charger une image locale' dialog is open on the right, providing instructions and a terminal snippet with the following commands:

```
az account set --subscription "Azure for Students"
az aks update -n aks-cluster -g MyRGroupe --attach-acr akscluster002
az acr login -n akscluster002
```

Below the dialog, two terminal windows show the execution of these commands. The first terminal shows the output of 'az aks update --attach-acr', and the second terminal shows the output of 'az acr login'. The second terminal also shows the output of 'docker images' and 'docker push' commands, indicating that the image has been successfully pushed to the registry.

Figure 5.25: Méthode deux

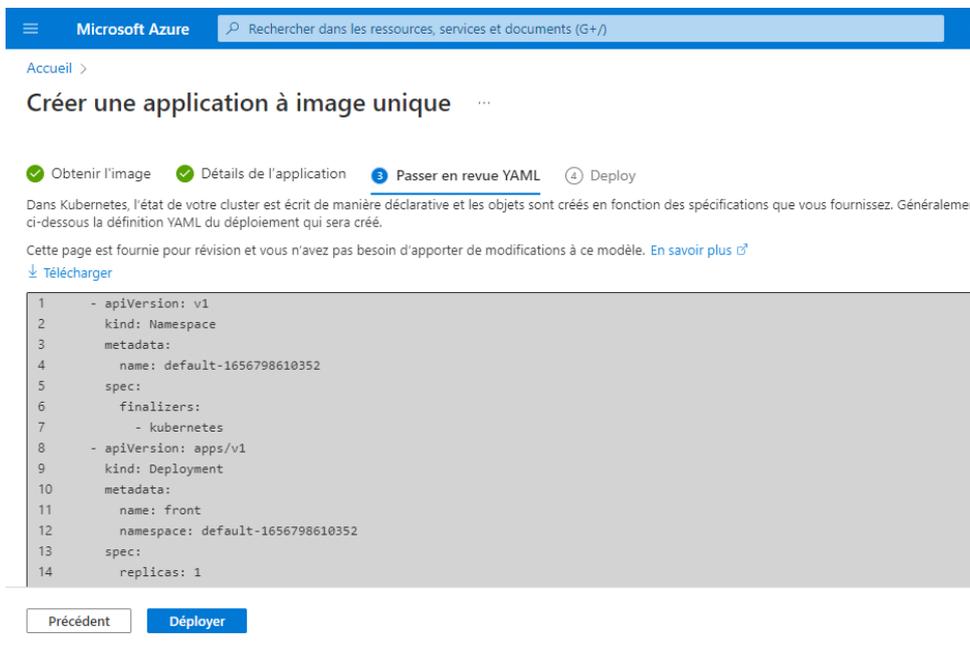
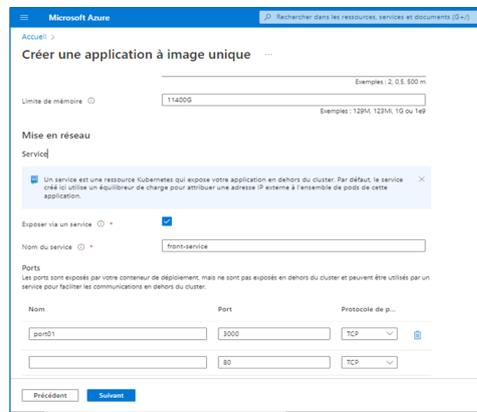
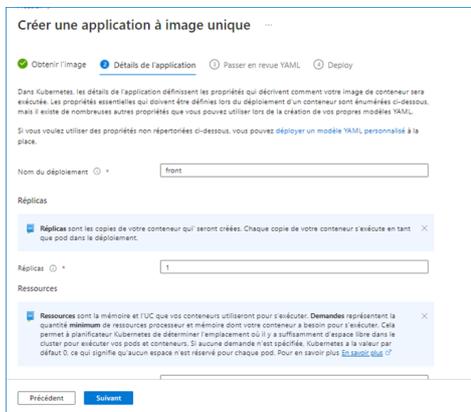
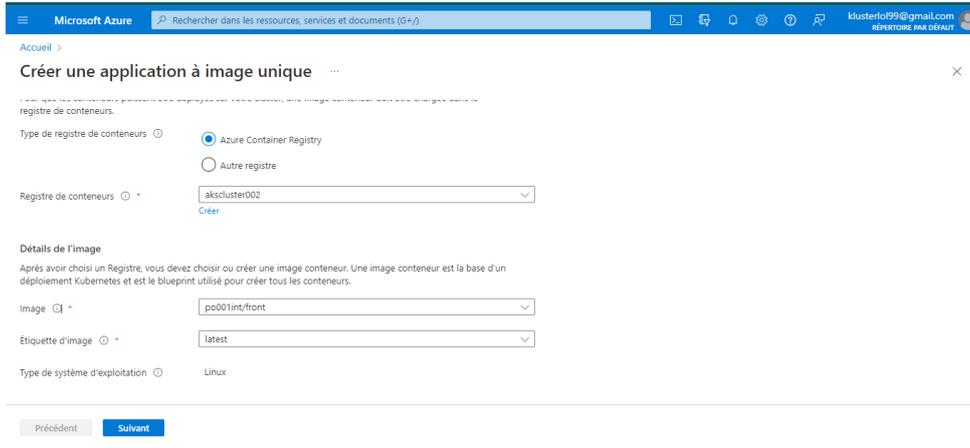


Figure 5.26: Méthode deux étapes

Accueil >

Créer une application à image unique

✓ Obtenir l'image
✓ Détails de l'application
✓ Passer en revue YAML
4 Deploy

✓ Votre application est en cours de déploiement. Merci de patienter.

Vous pouvez voir le déploiement que vous avez créé dans la liste ci-dessous et cliquer sur le nom du déploiement pour voir plus de détails, notamment les pods individuels créés dans le cadre du déploiement.

- 1. Connexion de votre Azure Container Registry**
La connexion de votre Azure Container Registry à votre cluster donne à votre cluster les autorisations dont il a besoin pour extraire des images de ce Registre.
 - akscluster002
- 2. Déploiement des ressources**
Vous pouvez voir le déploiement que vous avez créé dans la liste ci-dessous et cliquer sur le nom du déploiement pour voir plus de détails, notamment les pods individuels créés dans le c

Ressource	Type	État
default-1656798610352	Namespace	Success
front	Deployment	Success
front-service	Service	Success
front-pods	Pod	0/1 pods prêts

Précédent Fermer

Microsoft Azure Rechercher dans les ressources, services et documents (G+)

Accueil >

Créer une application à image unique

front-pods Pod 1/1 pods prêts

- 3. Étapes suivantes**
Voici quelques actions que vous pouvez effectuer une fois votre application déployée.
 - Afficher l'application**
Affichez l'application déployée en accédant à l'adresse IP externe associée au service front-end.
Afficher l'application
 - Découvrir comment créer et déployer des images conteneurs personnalisés**
Créez une configuration de pipeline GitOps pour déployer automatiquement votre application à partir du contrôle de code source sur votre cluster. Configurer GitOps
 - Envoyer des commentaires**
Contribuer à l'amélioration de cette page

Précédent Fermer

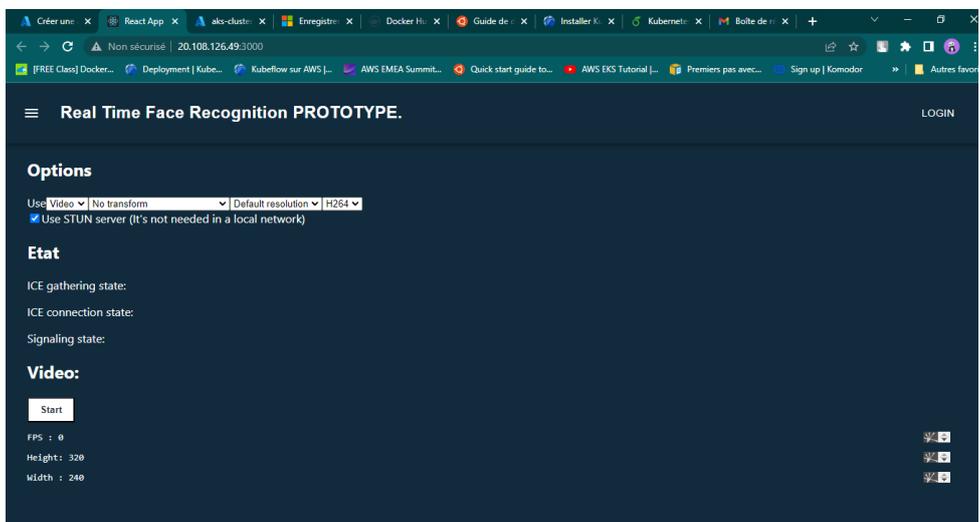


Figure 5.27: Déployer le front

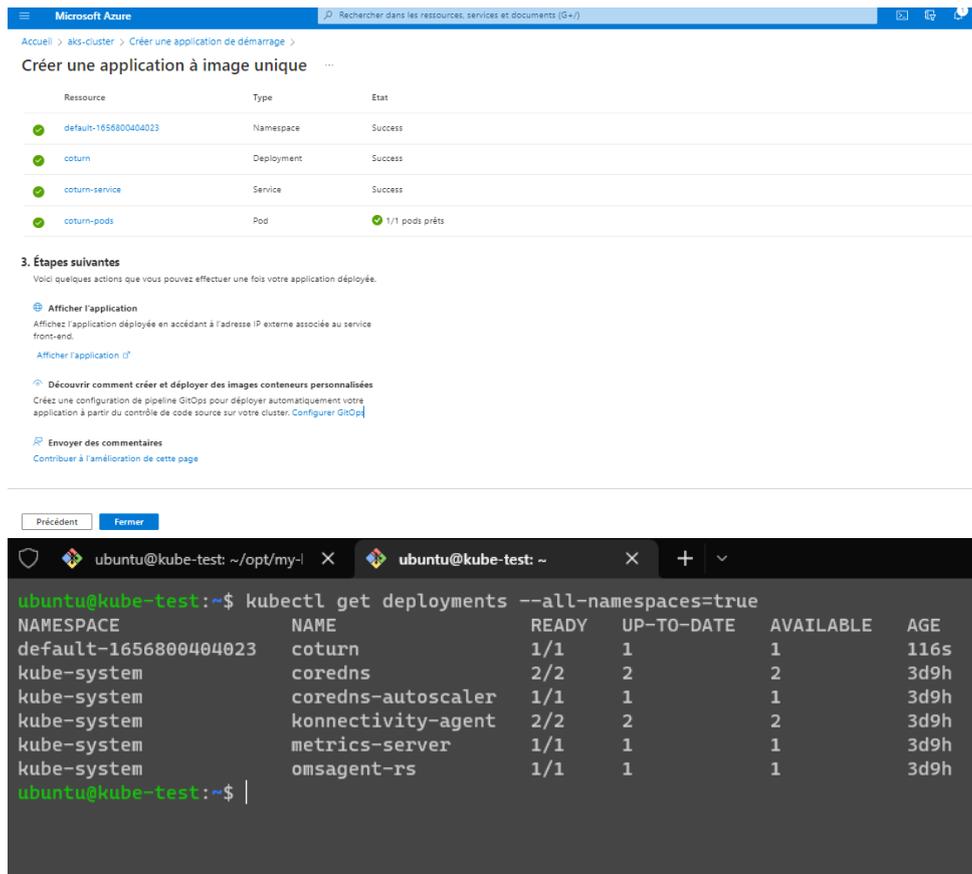


Figure 5.28: Déployer le coturn

3. Déploiement de kubeflow

Nous aurons besoin de kfctl pour exécuter les commandes qui contrôlent le déploiement. Après la connexion à Azure, nous allons installer kfctl puis suivre les instructions suivantes. [33]

```

# The following command is optional. It adds the kfctl binary to your path.
# If you don't add kfctl to your path, you must use the full path
# each time you run kfctl.
# Use only alphanumeric characters or - in the directory name.
export PATH=$PATH:<path-to-kfctl>

# Set KF_NAME to the name of your Kubeflow deployment. You also use this
# value as directory name when creating your configuration directory.
# For example, your deployment name can be 'my-kubeflow' or 'kf-test'.
export KF_NAME=<your choice of name for the Kubeflow deployment>

# Set the path to the base directory where you want to store one or more
# Kubeflow deployments. For example, /opt/.
# Then set the Kubeflow application directory for this deployment.
export BASE_DIR=<path to a base directory>
export KF_DIR=${BASE_DIR}/${KF_NAME}

# Set the configuration file to use when deploying Kubeflow.
# The following configuration installs Istio by default. Comment out
# the Istio components in the config file to skip Istio installation.
# See https://github.com/kubeflow/kubeflow/pull/3663
export CONFIG_URI="https://raw.githubusercontent.com/kubeflow/manifests/v1.2-branch/kfdef/kfctl_k8s_ist

mkdir -p ${KF_DIR}
cd ${KF_DIR}
kfctl apply -V -f ${CONFIG_URI}

```

Figure 5.29: Etapes du déploiement de kubeflow sur azure

```

ubuntu@k8s-test: ~/opt/my
Merged "aks-cluster" as current context in /home/ubuntu/.kube/config
ubuntu@k8s-test: ~$ export PATH=$PATH:/home/ubuntu
ubuntu@k8s-test: ~$ export KF_NAME=my-kubeflow
ubuntu@k8s-test: ~$ export BASE_DIR=opt
ubuntu@k8s-test: ~$ export KF_DIR=${BASE_DIR}/${KF_NAME}
ubuntu@k8s-test: ~$ export CONFIG_URI="https://raw.githubusercontent.com/kubeflow/manifests/v1.2-branch/kfdef/kfctl_k8s_istio.v1.2.0.yaml"
ubuntu@k8s-test: ~$ mkdir -p ${KF_DIR}
ubuntu@k8s-test: ~$ cd ${KF_DIR}
ubuntu@k8s-test: ~/opt/my-kubeflow$ kfctl apply -V -f ${CONFIG_URI}
INFO[0000] Downloading https://raw.githubusercontent.com/kubeflow/manifests/v1.2-branch/kfdef/kfctl_k8s_istio.v1.2.0.yaml to /tmp/162972702/tmp.yaml
filename="utils/k8utils.go:178"
INFO[0000] Downloading https://raw.githubusercontent.com/kubeflow/manifests/v1.2-branch/kfdef/kfctl_k8s_istio.v1.2.0.yaml to /tmp/555818725/tmp_app.yaml
filename="loaders/loaders.go:71"
INFO[0000] App directory /home/ubuntu/opt/my-kubeflow already exists filename="coordinator/coordinator.go:270"
INFO[0000] Writing KDef to kfctl_k8s_istio.v1.2.0.yaml filename="coordinator/coordinator.go:273"
INFO[0000] No name specified in KDef.Metadata.Name; defaulting to my-kubeflow based on location of config file: /home/ubuntu/opt/my-kubeflow/kfctl_k8s_istio.v1.2.0.yaml. filename="coordinator/coordinator.go:202"
INFO[0000]
*****
Notice anonymous usage reporting enabled using spartakus
To disable it
If you have already deployed it run the following commands:
cd $(pwd)
kubectl -n ${K8S_NAMESPACE} delete deploy -l app=spartakus
For more info: https://www.kubeflow.org/docs/other-guides/usage-reporting/
*****
filename="coordinator/coordinator.go:128"
INFO[0000] Creating directory /home/ubuntu/opt/my-kubeflow/.cache filename="kfconfig/types.go:450"
INFO[0000] Fetching https://github.com/kubeflow/manifests/archive/v1.2.0.tar.gz to /home/ubuntu/opt/my-kubeflow/.cache/manifests filename="kfconfig/types.go:409"
INFO[0000] Updating localPath to /home/ubuntu/opt/my-kubeflow/.cache/manifests/manifests-1.2.0 filename="kfconfig/types.go:569"
INFO[0000] Fetch succeeded; LocalPath /home/ubuntu/opt/my-kubeflow/.cache/manifests/manifests-1.2.0 filename="kfconfig/types.go:598"
INFO[0000] Processing application: namespaces filename="kustomize/kustomize.go:569"

```

```

ubuntu@k8s-test: ~/opt/my
INFO[0000] Processing application: metacontroller filename="kustomize/kustomize.go:569"
INFO[0000] Creating folder /home/ubuntu/opt/my-kubeflow/kustomize/metacontroller filename="kustomize/kustomize.go:667"
INFO[0000] Processing application: bootstrap filename="kustomize/kustomize.go:569"
INFO[0000] Creating folder /home/ubuntu/opt/my-kubeflow/kustomize/bootstrap filename="kustomize/kustomize.go:667"
INFO[0000] Processing application: spark-operator filename="kustomize/kustomize.go:569"
INFO[0000] Creating folder /home/ubuntu/opt/my-kubeflow/kustomize/spark-operator filename="kustomize/kustomize.go:667"
INFO[0000] Processing application: kubeflow-apps filename="kustomize/kustomize.go:569"
INFO[0000] Creating folder /home/ubuntu/opt/my-kubeflow/kustomize/kubeflow-apps filename="kustomize/kustomize.go:667"
INFO[0000] Processing application: knative filename="kustomize/kustomize.go:569"
INFO[0000] Creating folder /home/ubuntu/opt/my-kubeflow/kustomize/knative filename="kustomize/kustomize.go:667"
INFO[0000] Processing application: kfserving filename="kustomize/kustomize.go:569"
INFO[0000] Creating folder /home/ubuntu/opt/my-kubeflow/kustomize/kfserving filename="kustomize/kustomize.go:667"
INFO[0000] Processing application: spartakus filename="kustomize/kustomize.go:569"
INFO[0000] Creating folder /home/ubuntu/opt/my-kubeflow/kustomize/spartakus filename="kustomize/kustomize.go:667"
INFO[0000] /home/ubuntu/opt/my-kubeflow/.cache/manifests exists; not resyncing filename="kfconfig/types.go:473"
INFO[0000] namespace: kubeflow filename="utils/k8utils.go:433"
INFO[0000] Log cluster name into KDef: aks-cluster filename="kustomize/kustomize.go:253"
INFO[0000] Deploying application namespaces filename="kustomize/kustomize.go:266"
namespace/cert-manager unchanged
namespace/kubeflow unchanged
INFO[0000] Successfully applied application namespaces filename="kustomize/kustomize.go:291"
INFO[0000] Deploying application application filename="kustomize/kustomize.go:266"
serviceaccount/application-controller-service-account unchanged
clusterrole.rbac.authorization.k8s.io/application-controller-cluster-role unchanged
clusterrolebinding.rbac.authorization.k8s.io/application-controller-cluster-role-binding unchanged
service/application-controller-service unchanged
statefulset.apps/application-controller-stateful-set configured
WARN[0000] Encountered error applying application application: (Kubeflow.error): Code 500 with message: Apply.Run : [unable to recognize "/tmp/kout398799676": no matches for kind "CustomResourceDefinition" in version "apiextensions.k8s.io/v1beta1", unable to recognize "/tmp/kout398799676": no matches for kind "Application" in version "app.k8s.io/v1beta1"] filename="kustomize/kustomize.go:284"
WARN[0000] Will retry in 2 seconds. filename="kustomize/kustomize.go:285"
serviceaccount/application-controller-service-account unchanged
clusterrole.rbac.authorization.k8s.io/application-controller-cluster-role unchanged
clusterrolebinding.rbac.authorization.k8s.io/application-controller-cluster-role-binding unchanged
service/application-controller-service unchanged

```

Figure 5.30: Déploiement de kubeflow sur azure

Conclusion Générale

Cette recherche a été menée en mettant l'accent sur les possibilités de mettre en production des modèles d'apprentissage automatique, Nous avons tout d'abord commencé par la mise en œuvre de docker, une technologie de conteneurisation conçue pour exécuter un service ou une application unique de manière redondantes, ainsi que Kubernetes conçu à l'origine pour orchestrer des applications de microservices à grande échelle.

Nous avons ensuite testé le déploiement local avec un seul nœud à l'aide de l'outil Minikube, ce qui nous a permis de tester les images officielles extraites du registre central de Docker appelé Docker Hub ainsi que les microservices de notre système. Cependant, nous avons rencontré de nombreux problèmes que nous avons pu traiter par la suite comme : Dockershim, Imagepullbackoff, Installation...

Ensuite, il y a le déploiement cloud que nous avons effectué sur les plates-formes cloud dominantes : AWS et Azure, Nous avons ainsi découvert les conditions de fonctionnement et de production de différents services dans le cloud. Le résultat parvenue de ce travail est la mise en place de kubeflow, une solution conçue pour développer des applications de machine learning et déployer sur Kubernetes.

Toutes les étapes franchies au cours de ce travail nous a permis de découvrir et de développer des concepts liés aux technologies les plus courantes, d'essayer certaines possibilités, de déboguer certains problèmes, Et pour finalement arriver à une solution qui n'est peut-être que le début d'un autre problème qui peut continuer à améliorer le développement de notre travail.

Bibliography

- [1] <https://venturebeat.com/2019/07/19/why-do-87-of-data-science-projects-never-make-it-into-production/>.
- [2] <https://www.seldon.io/how-to-deploy-your-machine-learning-models>.
- [3] [https://fr.wikipedia.org/wiki/Cycle_{nV}](https://fr.wikipedia.org/wiki/Cycle_nV).
- [4] <https://www.edureka.co/blog/devops-lifecycle/>.
- [5] <https://milannovic.medium.com/robot-framework-ci-cd-with-azure-devops-cf708a64b389>.
- [6] <http://www.https://codefresh.io/learn/software-deployment/section-software-deployment-strategiesl>.
- [7] <https://www.fiddler.ai/mlops>.
- [8] <https://databricks.com/glossary/mlops>.
- [9] ².
- [10] <https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained>.
- [11] <https://developer.nvidia.com/blog/choosing-a-server-for-deep-learning-training/>.
- [12] <https://www.lebigdata.fr/gpu-ia-big-data>.
- [13] <https://scikit-learn.org/stable/modules/clustering.html>.
- [14] https://fr.ryte.com/wiki/Release_mangement.
- [15] <https://microservices.io/>.
- [16] <https://www.lebigdata.fr/reconnaissance-faciale-tout-savoir>.
- [17] <https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/benefits>.
- [18] <https://www.stackscale.com/blog/cloud-service-models/>.
- [19] <https://datascientest.com/docker-guide-complet>.
- [20] <https://www.sdxcentral.com/cloud/containers/definitions/containers-vs-vms/>.
- [21] <https://datascientest.com/docker-guide-complet>.
- [22] <https://kubernetes.io/fr/docs/concepts/overview/what-is-kubernetes/>.

- [23] <https://www.journaldunet.fr/web-tech/guide-de-l-entreprise-digitale/1440612-kubernetes-definition-architecture-conseil/>.
- [24] <https://wiki.sfeir.com/kubernetes/architecture/composants/services/>.
- [25] <https://www.seldon.io/deploying-machine-learning-models-on-kubernetes#:text=Instead%20of%20different%20parts%20of,on%20local%20or%20cloud%20servers.>
- [26] <https://platform9.com/blog/why-and-how-to-run-machine-learning-workloads-on-kubernetes/>.
- [27] <https://ubuntu.com/ai/what-is-kubeflow>.
- [28] <https://blent.ai/kubernetes-et-machine-learning/>.
- [29] <https://github.com/RumbleFiend/recog-coturn>.
- [30] <https://github.com/RumbleFiend/recog-front>.
- [31] https://github.com/RumbleFiend/recog_prototype.
- [32] https://docs.aws.amazon.com/fr_fr/eks/latest/userguide/what-is-eks.html.
- [33] <https://www.kubeflow.org/docs/distributions/azure/deploy/install-kubeflow/>.