Ministère de l'enseignement
supérieur et de la recherche
scientifique
*** * ***
Université de Carthage
*** * ***
**Institut National
des Sciences Appliquées et de
Technologie**

Projet de fin d'année

Filière : Réseaux informatiques et télécommunications

Niveau : 4ème année

Sujet :

Déploiement Avancé sur Kubernetes

Préparé par :

Nermine BACHA

Achraf HABIB

Khalil EL ATI

Chedi GHRIBI

Encadrante :

Mme Saloua BEN YAHIA

Année universitaire : 2023/2024

Table des matières

Table des figures

Remerciements

Ce travail n'aurait pas pu être réalisé sans les contributions précieuses de nombreuses personnes, à qui nous espérons exprimer notre gratitude à travers ces lignes.

Nos remerciements s'adressent tout d'abord à Dr. Saloua Ben Yahia, qui nous a encadrés tout au long de ce projet. C'est grâce à ses conseils avisés, sa guidance précieuse et son soutien que nous avons pu finaliser ce projet.

Nous tenons également à exprimer notre sincère gratitude à Dr. Sofiene Ouni, membre éminent du jury, pour avoir gracieusement accepté de revoir notre projet. En tant que professeur distingué à l'Institut National des Sciences Appliquées et de Technologie (INSAT), son expertise enrichira notre processus d'évaluation.

Enfin, nous souhaitons remercier toutes les personnes qui ont suivi les péripéties de ce projet, ainsi que nos familles et amis.

Introduction Générale

Dans un monde où la technologie et l'innovation avancent à un rythme effréné, les entreprises sont constamment confrontées au défi de rester compétitives. L'une des clés pour y parvenir réside dans la capacité à déployer des mises à jour de logiciels de manière rapide, efficace et fiable. La fréquence des mises à jour, la réduction des temps d'arrêt et l'optimisation des coûts sont des facteurs cruciaux pour assurer une qualité de service continue et satisfaire les attentes croissantes des utilisateurs.
Au fil des ans, les méthodes de déploiement ont évolué pour répondre à ces besoins. Les approches traditionnelles, souvent manuelles et sujettes à des erreurs, ont laissé place à des processus automatisés et sophistiqués. Cette évolution est en grande partie due à l'émergence de nouvelles technologies et de nouvelles pratiques de déploiement. Les entreprises cherchent désormais à implémenter des stratégies de déploiement qui non seulement réduisent les risques d'interruption de service mais aussi permettent des mises à jour fréquentes et transparentes.

Cependant, la mise en place de ces stratégies de déploiement avancées pose plusieurs défis. Les entreprises doivent faire face à des choix complexes concernant la méthode la plus adaptée pour leurs environnements spécifiques. Chaque stratégie de déploiement a ses propres avantages et inconvénients en termes de coût, de complexité, de risque et d'impact sur la disponibilité des services. La question centrale qui se pose est donc : comment les entreprises peuvent-elles choisir la meilleure stratégie de déploiement pour répondre à leurs besoins de fréquence élevée de mises à jour, tout en minimisant les coûts et en évitant les interruptions de service ?

La motivation derrière ce projet est née de notre passion pour les technologies cloud et les systèmes distribués ce qui nous a naturellement conduits à explorer des solutions innovantes pour améliorer les processus de déploiement. En explorant les avantages et les inconvénients des différentes stratégies, ce projet nous permet également de développer nos compétences techniques et de renforcer notre compréhension des défis réels auxquels sont confrontées les entreprises. Nous espérons que nos travaux pourront non seulement nous enrichir académiquement mais aussi apporter une contribution significative à la communauté des développeurs et aux entreprises.

Dans ce contexte, notre projet vise à apporter des contributions significatives dans le domaine des déploiements d'applications et à établir une étude comparative entre les différentes stratégies de déploiement avancées sur Kubernetes. Tout en intégrant les pratiques DevOps et l'Infrastructure as Code (IaC), et en fournissant des recommandations pratiques pour améliorer les processus de déploiement d'applications.

Chapitre 1

Contexte du Projet

1.1 Introduction

Dans le contexte actuel de développement logiciel, les approches traditionnelles ne suffisent plus à répondre aux exigences de rapidité et de qualité. Les pratiques modernes telles que DevOps, Kubernetes et Infrastructure as Code (IaC) jouent un rôle central dans la transformation des processus de développement et de déploiement des applications. Ce chapitre a pour objectif d'introduire ces concepts fondamentaux qui sont essentiels pour comprendre les stratégies avancées de déploiement abordées dans ce projet.

1.2 Concepts de base

1.2.1 DevOps

Le mouvement DevOps unifie les processus de développement logiciel et les opérations IT, favorisant une meilleure collaboration entre les équipes. Cela permet non seulement de raccourcir les cycles de développement, mais aussi d'assurer des déploiements plus stables et plus prévisibles. L'automatisation des processus, la livraison continue, et la surveillance proactive sont des aspects clés de DevOps qui permettent d'optimiser les déploiements fréquents.
Dans le cadre de ce projet, l'adoption de pratiques DevOps est cruciale pour créer une pipeline CI/CD efficace. Cela nous permettra d'automatiser les tests et les déploiements, réduisant ainsi les erreurs humaines et accélérant le processus de mise en production. En intégrant DevOps, nous pouvons également surveiller en continu les performances et la santé de nos applications, garantissant ainsi une réponse rapide aux problèmes éventuels.

1.2.2 Kubernetes

Kubernetes, en tant que plateforme d'orchestration de conteneurs, est indispensable pour gérer les applications conteneurisées dans ce projet. Il offre une infrastructure robuste et flexible qui simplifie le déploiement, la gestion et la mise à l'échelle des applications. Les différentes stratégies de déploiement qu'il propose, telles que le rolling update, le blue-green deployment, et le canary deployment, permettent de minimiser les interruptions de service et de gérer efficacement les risques associés aux mises à jour.
L'utilisation de Kubernetes dans ce projet est essentielle pour garantir une gestion efficace des conteneurs et une mise à l'échelle facile des applications. En exploitant ses capacités avancées, nous pouvons déployer des applications de manière fiable et cohérente, tout en assurant leur disponibilité et leur performance.

1.2.3 Infrastructure as Code (IaC)

L'Infrastructure as Code (IaC) est une pratique clé qui permet de gérer et provisionner les ressources informatiques à l'aide de fichiers de configuration lisibles par des machines. Dans le cadre de ce projet, IaC est crucial pour automatiser la configuration de l'infrastructure, assurer la cohérence et réduire les risques d'erreurs humaines. En traitant l'infrastructure de la même manière que le code logiciel, nous pouvons versionner, suivre les modifications et reproduire les environnements de manière fiable.
IaC nous permet également de répondre rapidement aux besoins croissants des applications en facilitant la mise à l'échelle de l'infrastructure. En utilisant des outils d'IaC, nous pouvons créer et gérer l'infrastructure nécessaire pour notre pipeline CI/CD et nos déploiements sur Kubernetes, tout en garantissant une gestion efficace et une grande flexibilité.

1.3 Conclusion

A travers ce chapitre, nous avons présenté les concepts clés de DevOps, Kubernetes et Infrastructure as Code (IaC), essentiels pour améliorer les processus de développement et de déploiement. DevOps permet des cycles de développement plus rapides et stables, Kubernetes gère efficacement les applications conteneurisées, et IaC automatise la configuration de l'infrastructure. Forts de ces bases, nous allons maintenant explorer les stratégies de déploiement sur Kubernetes dans le chapitre suivant.

Chapitre 2

Kubernetes et les Stratégies de Déploiement

2.1 Introduction

Dans ce chapitre, nous aborderons Kubernetes, une plateforme d'orchestration de conteneurs largement utilisée, et les diverses stratégies de déploiement el le concept de Rolling qu'elle propose.

2.1.1 Définition et Composants de Kubernetes

2.1.1.1 Définition

Kubernetes est un moteur d'orchestration de conteneur open source permettant d'automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Le projet open source est hébergé par la Cloud Native Computing Foundation. (CNCF). [1]

2.1.1.2 Composants de Kubernetes

— **Pods** :
Un Pod en Kubernetes est une abstraction au-dessus du conteneur. Cela signifie que plutôt de gérer des conteneurs individuellement, Kubernetes utilise des Pods comme unité de base pour le déploiement et la gestion des conteneurs.
Un Pod suit un cycle de vie défini, démarrant de l'état Pending (en attente), passant à Running (en cours d'exécution) une fois que toutes ses ressources sont disponibles et qu'il a été planifié sur un nœud, et finalement se terminant dans les états Succeeded (réussi) ou Failed (échoué).

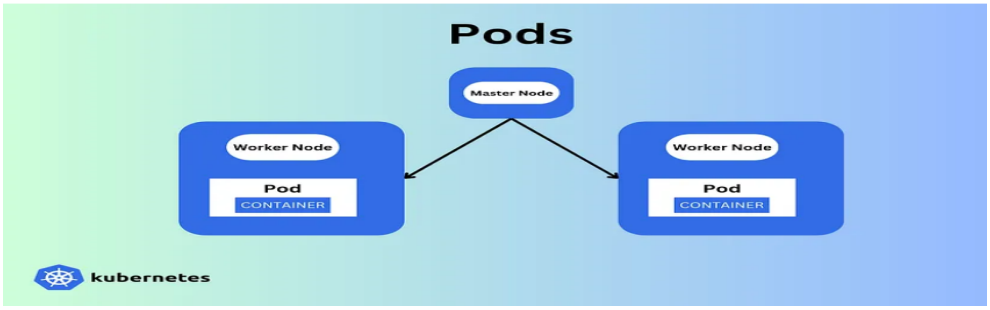

FIGURE 2.1 – Pods sur Kubernetes

— **Deployment** :
Un Deployment crée et gère un ensemble de réplicas de Pods, assurant ainsi que le nombre spécifié de Pods est toujours en cours d'exécution.
Si un Pod tombe en panne ou devient non fonctionnel, le Deployment crée automatiquement un nouveau Pod pour maintenir le nombre de réplicas désiré. On peut facilement scaler un Deployment en augmentant ou diminuant le nombre de réplicas

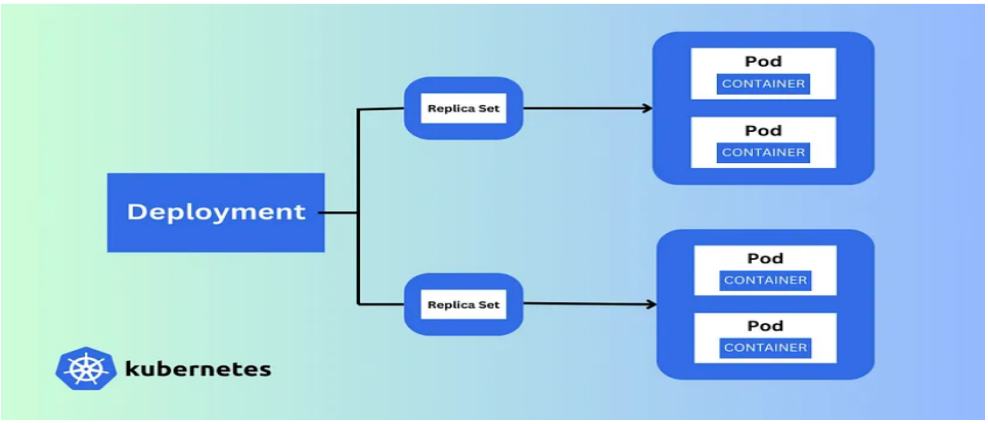

FIGURE 2.2 – Deployment sur Kubernetes

— **Volume** :
Les volumes sont utilisés pour fournir un espace de stockage aux conteneurs d'un Pod.
Kubernetes supporte plusieurs types de volumes, chacun avec ses propres caractéristiques et usages

— **emptyDir** : Un volume temporaire qui partage l'espace de stockage avec le nœud jusqu'à ce que le Pod soit supprimé.

— **hostPath** : Montage d'un fichier ou répertoire du système de fichiers de l'hôte à l'intérieur du Pod, souvent utilisé pour des tâches spécifiques de gestion du système.

— **nfs** : Un volume NFS (Network File System) qui monte un dossier partagé depuis un serveur NFS.

— **PersistentVolumeClaim (PVC)** : Utilise un objet PersistentVolume (PV) pour fournir un stockage persistant et dynamique basé sur le besoin.

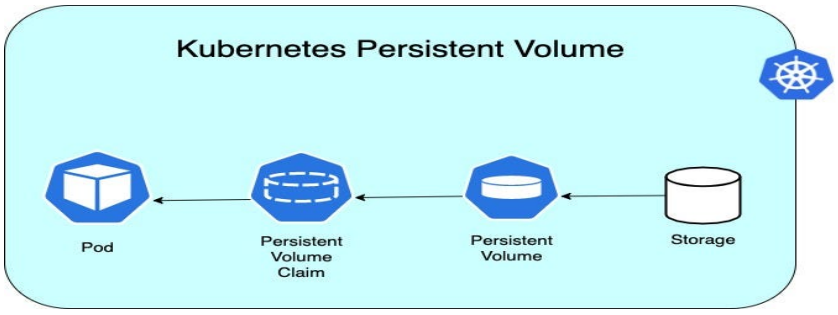

FIGURE 2.3 – Volume sur Kubernetes

— **Services**
un Service est une abstraction qui définit un ensemble logique de Pods et une politique d'accès à eux. Le principal but d'un Service est de permettre aux applications externes d'accéder aux services tournant dans un ensemble de Pods, indépendamment de la réplication et de la localisation des Pods dans le cluster. Le Service permet de fournir une adresse IP fixe et un nom DNS par lequel les Pods peuvent être accessibles, masquant ainsi lsa complexité de leur déploiement et leur gestion. Kubernetes offre plusieurs types de Services, chacun adapté à des usages spécifiques :

- **ClusterIP** : Expose le Service sur une adresse IP interne dans le cluster. C'est le type de Service par défaut et il est utile pour la communication interne au sein du cluster.
- **NodePort** : Expose le Service sur chaque adresse IP des nœuds à un port statique (le NodePort).
  Cela permet d'accéder au Service depuis l'extérieur du cluster en utilisant :
  <IP_Node> :<NodePort>
- **LoadBalancer** : Intègre les Services avec les load balancers disponibles chez un fournisseur de cloud. Cela attribue une adresse IP externe fixe au Service pour que les requêtes externes atteignent le Service.
- **ExternalName** : Mappe le Service à un nom DNS, et non à une sélection de Pods typique. Ceci est utilisé pour orienter le service vers un URI externe en dehors du cluster.

— **Ingress :**
un Ingress est une ressource qui gère l'accès externe aux services dans un cluster, typiquement des requêtes HTTP ou HTTPS. Il permet de router le trafic externe vers les services internes selon des règles définies. Cela simplifie la configuration des règles de routage et est essentiel pour des applications accessibles publiquement.

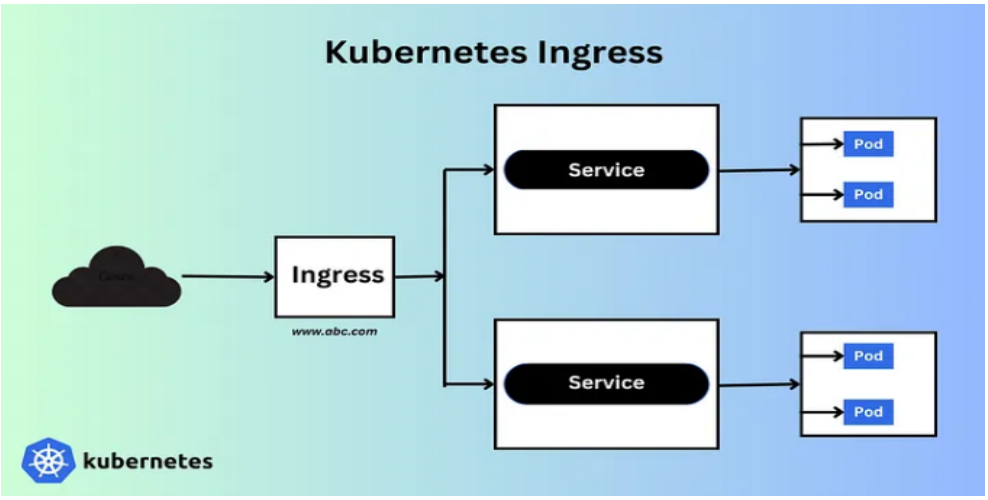

FIGURE 2.4 – Ingress sur kubernetes

— **ConfigMap :**
Config Map est essentiel pour la gestion des configurations dans Kubernetes. Une Config Map peut stocker des fichiers de configuration, des variables d'environnement, et d'autres données de configuration, permettant aux conteneurs de lire ces configurations au démarrage plutôt que de les inclure directement dans l'image du conteneur. Cela simplifie la gestion des environnements et des configurations sans nécessiter de reconstruire les images de conteneur, facilitant ainsi les mises à jour et les modifications de configuration.

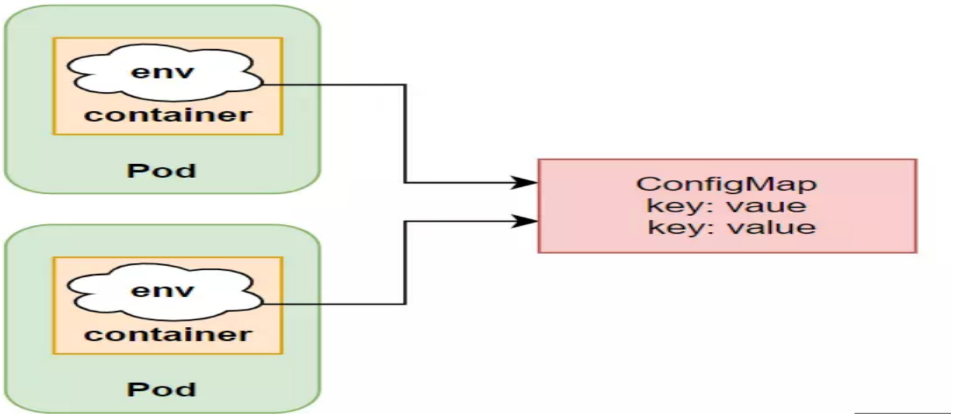

FIGURE 2.5 – ConfigMap sur Kubernetes

— **Secret :**
Secret joue un rôle crucial dans la gestion des informations sensibles au sein d'un cluster Kubernetes. Il est utilisé pour stocker des informations telles que les mots de passe, les jetons OAuth, et les clés API. Les données stockées dans un Secret sont chiffrées dans le stockage et transmises de manière sécurisée aux nœuds qui exécutent les Pods nécessitant l'accès à ces secrets. L'utilisation de Secrets aide à sécuriser l'accès aux outils, applications et services externes, réduisant ainsi le risque de fuites d'informations sensibles.

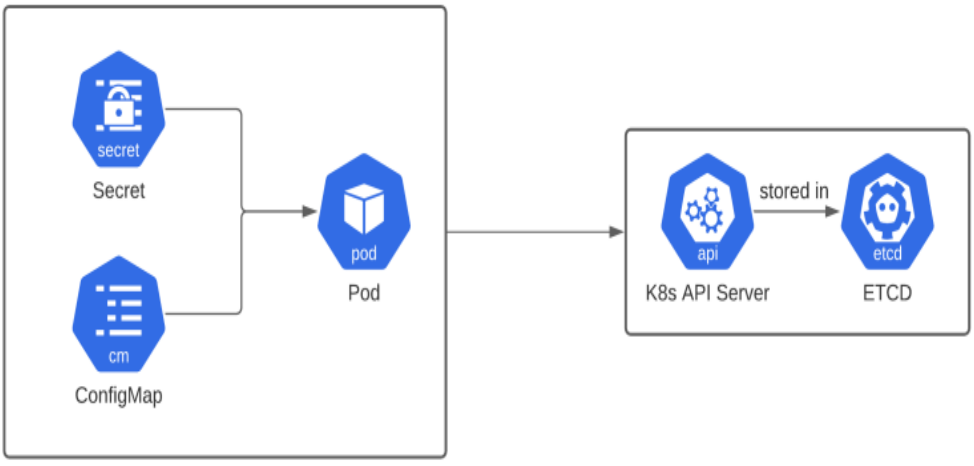

FIGURE 2.6 – Secret sur Kubernetes

2.1.1.3 Architecture de Kubernetes

Un cluster Kubernetes est constitué d'un ensemble de machines ouvrières, appelées « Nœuds ». Il existe deux types de nœuds dans un cluster Kubernetes :

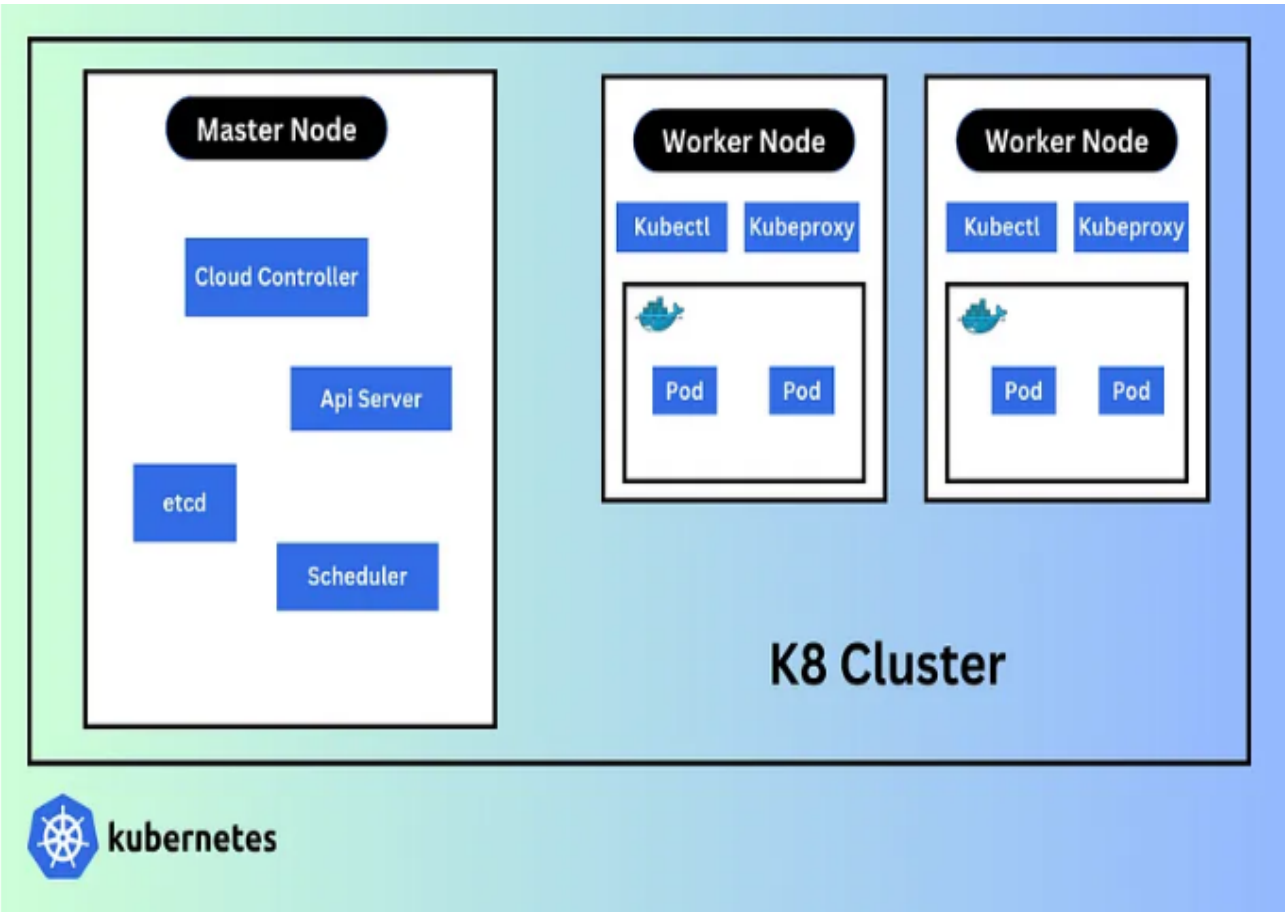

FIGURE 2.7 – Architecture de kubernetes

— **Master Node** : Le Master Node gère le cluster, il prend des décisions globales sur le cluster, et exécute plusieurs composants critiques :
  — **API Server :**
  Le Serveur APIest le cœur de la communication dans un cluster Kubernetes. Il traite toutes les requêtes, effectue des validations, et interagit avec le cluster etcd pour maintenir un état cohérent et actualisé du cluster. C'est par lui que toutes les commandes kubectl sont exécutées, facilitant ainsi la gestion des ressources du cluster.
  — **Controller Manager :**
  Le Gestionnaire de Contrôleurs joue un rôle clé dans la gestion de l'état du cluster. Il s'assure que l'état actuel du cluster correspond à l'état souhaité spécifié par l'utilisateur. Pour cela, il surveille en continu l'état du cluster et initie des actions correctives en cas de divergence.
  — **Scheduler :**
  Le Planificateur est responsable de l'assignation des Pods aux nœuds. Il sélectionne le nœud le plus approprié basé sur les exigences de ressources des Pods et la disponibilité des nœuds, garantissant ainsi une distribution équilibrée et efficace des charges de travail à travers le cluster.

- **Etcd :**
  Etcd est un magasin de données clé-valeur hautement disponible et distribué qui sert de registre de configuration et de state pour le cluster. Il stocke l'état du cluster, la configuration, les secrets, et d'autres données cruciales, faisant de lui la source de vérité pour le cluster.

— **Worker Node** : Les Worker Nodes sont les machines qui exécutent les applications et les workloads effectifs. Ils sont gérés par les Master Nodes

- **Kubelet** :
  Le **Kubelet** fonctionne comme un agent sur chaque nœud, assurant que les conteneurs opèrent dans les Pods conformément aux spécifications fournies par le serveur API. Il surveille continuellement l'état des Pods et gère activement les conteneurs pour garantir qu'ils fonctionnent de manière optimale et conforme aux attentes définies.
- **Kube Proxy** :
  Le **Kube Proxy** gère les règles de réseau sur les nœuds, facilitant ainsi la communication interne entre les Pods ainsi que leur interaction avec les services externes. De plus, il est essentiel pour maintenir la connectivité et la performance du réseau à travers tout le cluster, garantissant que le trafic réseau est correctement dirigé et optimisé pour la performance et la sécurité.
- **Interface du Runtime de Conteneur (CRI)** : L'**Interface du Runtime de Conteneur (CRI)** est le logiciel responsable de l'exécution des conteneurs. Elle gère les runtimes de conteneurs, tels que Docker, en chargeant les images de conteneurs et en supervisant la création et la gestion des conteneurs. Cette interface assure que les conteneurs fonctionnent correctement et sont maintenus à jour selon les besoins des applications déployées dans le cluster.

2.2 Stratégies de Déploiement

2.2.1 Rolling Update

La stratégie de déploiement Rolling sur Kubernetes est une méthode pour mettre à jour et déployer de nouvelles versions de logiciels de manière contrôlée et progressive. Au lieu de déployer les mises à jour en une seule fois.C'est la stratégie de déploiement par défaut dans Kubernetes. Kubernetes introduit les modifications de manière incrémentale en remplaçant les pods un par un de la version précédente de notre application par des pods de la nouvelle version , réduisant ainsi le risque de temps d'arrêt et permettant des retours en arrière faciles en cas d'erreurs.

Le déploiement Rolling implique la création d'un nouvel ensemble de réplicas avec la version mise à jour du logiciel tout en réduisant progressivement l'ensemble de réplicas ancien. Cela permet de déployer la nouvelle version tout en gardant l'ancienne version en fonctionnement, assurant ainsi qu'il n'y a pas d'interruption de service. Une fois la nouvelle version entièrement déployée, l'ancien ensemble de réplicas est supprimé, et le déploiement est terminé. La stratégie de déploiement Rolling est essentielle pour garantir une haute disponibilité et une fiabilité dans les systèmes distribués complexes.
Lors de l'utilisation de la stratégie RollingUpdate, il y a **deux options supplémentaires** qui nous permettent de 'fine tune' le processus de mise à jour :

— **maxSurge** : Le nombre de pods qui peuvent être créés au-delà du nombre souhaité de pods pendant une mise à jour. Cela peut être un nombre absolu ou un pourcentage du nombre de réplicas. La valeur par défaut est 25.

— **maxUnavailable** : Le nombre de pods qui peuvent être indisponibles pendant le processus de mise à jour. Cela peut être un nombre absolu ou un pourcentage du nombre de réplicas, la valeur par défaut est 25.

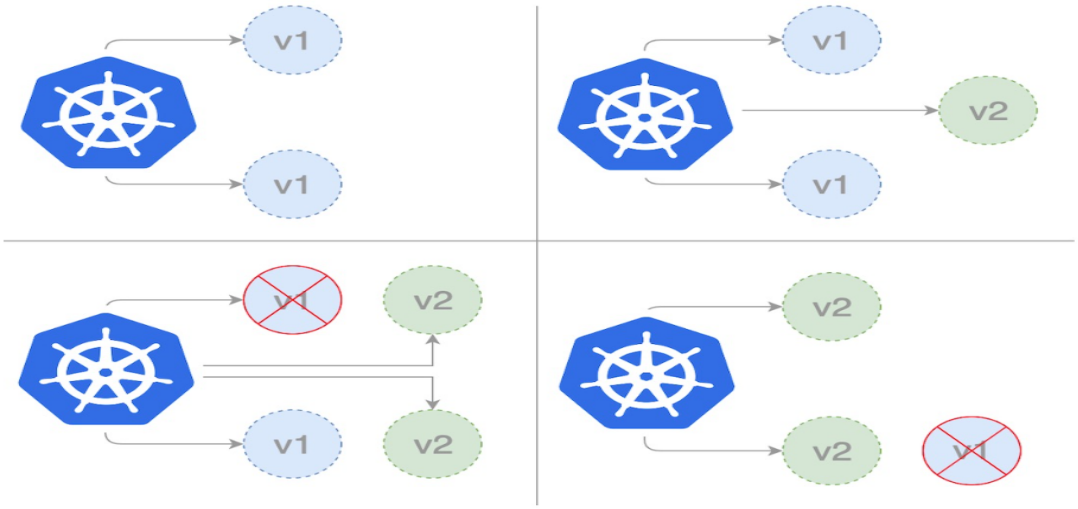

FIGURE 2.8 – Rolling Update

Avantages

— **Haute Disponibilité** : En introduisant les mises à jour de manière incrémentale, le déploiement Rolling assure que le service reste disponible pendant tout le processus de mise à jour.

— **Réduction des Risques** : La mise à jour progressive permet de minimiser les risques d'erreurs affectant l'ensemble du système, facilitant la détection précoce des problèmes.

— **Rollback Facile** : En cas de détection d'une erreur, il est facile de revenir en arrière aux versions précédentes, ce qui assure la stabilité du système.

— **Aucune Interruption de Service** : Le service reste continu et ininterrompu pendant le déploiement, car l'ancienne version reste active jusqu'à ce que la nouvelle version soit complètement opérationnelle.

— **Flexibilité et Contrôle** : Le déploiement Rolling offre un contrôle granulaire sur le processus de mise à jour, permettant d'ajuster le rythme de déploiement en fonction des besoins et des conditions.

2.2.2 Recreate

La stratégie de déploiement Recreate dans Kubernetes fonctionne en supprimant tous les pods de la version précédente de l'application avant de créer des pods de la nouvelle version. Cela signifie qu'il y a une période d'indisponibilité lorsque les anciens pods sont supprimés et avant que les nouveaux pods ne soient entièrement opérationnels. Cette stratégie est utile lorsque la nouvelle version de l'application n'est pas compatible avec l'ancienne, ou lorsque vous voulez éviter les complications liées à l'exécution de deux versions différentes de l'application en même temps.

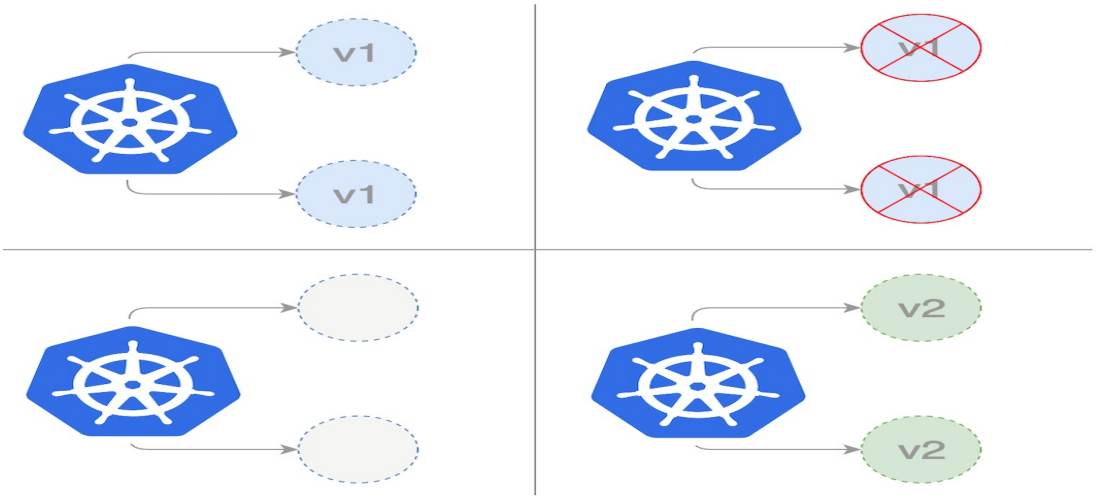

FIGURE 2.9 – Recreate

Avantages

— **Simplicité** : Le processus de déploiement est simple car il ne nécessite pas de gestion simultanée de différentes versions de l'application.

— **Cohérence des versions** : Il n'y a jamais de chevauchement entre les versions de l'application, ce qui élimine les problèmes de compatibilité entre versions.

2.2.3 Blue-Green Deployment

La stratégie de déploiement Blue-Green sur Kubernetes est une technique utilisée pour déployer de nouvelles versions d'une application tout en minimisant les temps d'arrêt et les risques associés. Cette méthode consiste à maintenir deux environnements identiques : l'un servant d'environnement de production actif **bleu** et l'autre comme candidat pour la nouvelle version **vert**. Avant de basculer vers l'environnement de production, le candidat est minutieusement testé pour s'assurer qu'il fonctionne correctement.

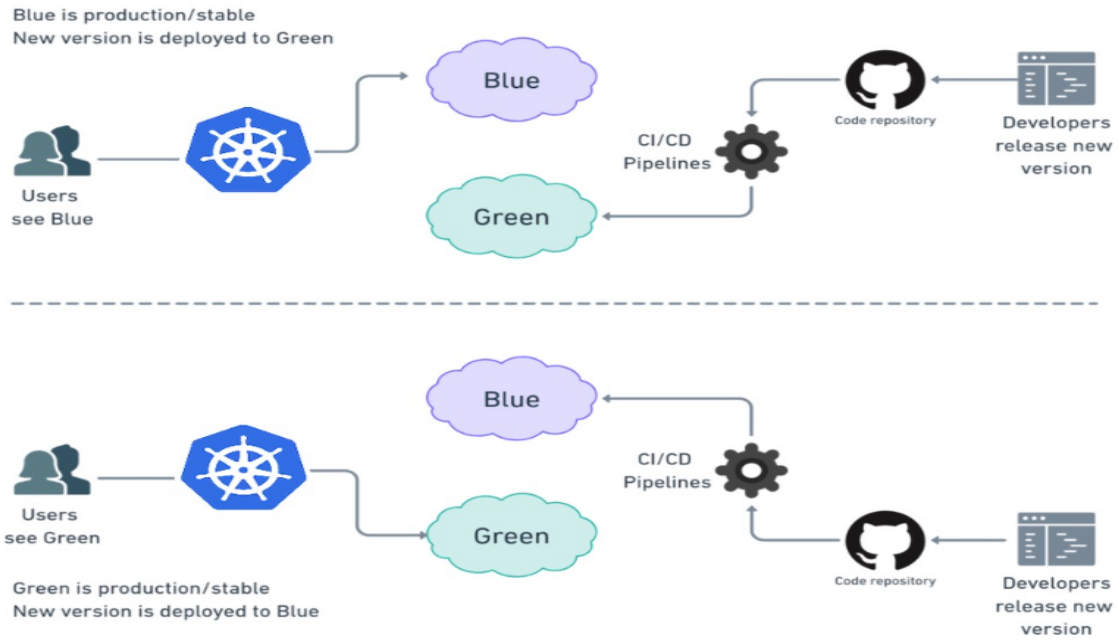

FIGURE 2.10 – Blue-Green

Avantages

- **Minimisation des Temps d'Arrêt** : La transition entre les environnements bleu et vert se fait sans interruption de service, garantissant ainsi une disponibilité continue pour les utilisateurs.
- **Réduction des Risques** : En testant la nouvelle version dans l'environnement vert avant de la passer en production, les risques de défaillances ou d'erreurs sont significativement réduits.
- **Reversibilité Facile** : Si un problème survient après le passage à la nouvelle version, il est facile de revenir rapidement à l'environnement bleu, garantissant ainsi la stabilité du service.
- **Amélioration de la Qualité** : Cette méthode permet de tester en conditions réelles la nouvelle version de l'application, améliorant ainsi la qualité et la fiabilité des déploiements.

2.2.4 Canary Deployment

La stratégie de déploiement Canary sur Kubernetes est une technique utilisée pour déployer de nouvelles fonctionnalités ou modifications à un petit sous-ensemble d'utilisateurs ou de serveurs avant de déployer la mise à jour sur l'ensemble du système. Cela se fait en créant un nouvel ensemble de réplicas avec la version mise à jour du logiciel tout en maintenant l'ensemble de réplicas original en fonctionnement. Un petit pourcentage du trafic est ensuite dirigé vers le nouvel ensemble de réplicas, tandis que la majorité du trafic continue d'être servi par l'ensemble de réplicas original. Cela permet de

tester la nouvelle version dans un environnement réel tout en minimisant le risque de problèmes affectant l'ensemble du système. Si des problèmes sont détectés lors du déploiement canary, il est possible de revenir rapidement à l'ensemble de réplicas original. L'image ci-dessous montre la manière la plus simple et directe de réaliser un déploiement canari. Une nouvelle version est déployée sur un sous-ensemble de serveurs. Pendant ce temps, nous observons comment les machines mises à jour se comportent. Nous

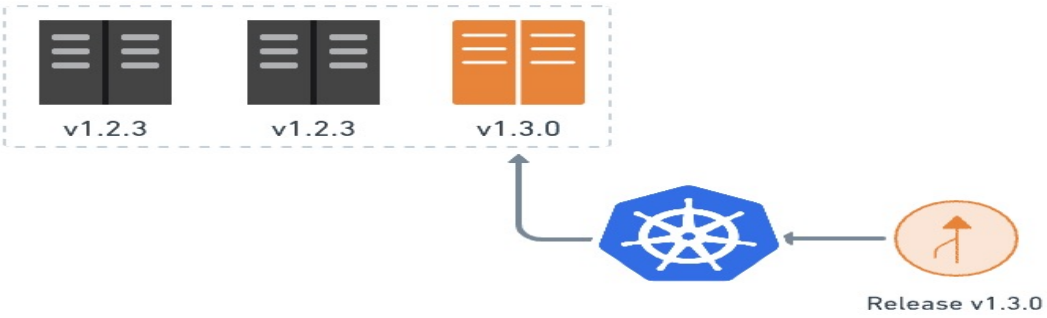

FIGURE 2.11 – Canary

vérifions les erreurs et les problèmes de performance, et nous écoutons les retours des utilisateurs. Au fur et à mesure que nous gagnons en confiance dans le canari, nous continuons à l'installer sur le reste des machines jusqu'à ce qu'elles exécutent toutes la dernière version

FIGURE 2.12 – Canary

Avantages

- **Minimisation des Risques** : En limitant le déploiement initial à un petit pourcentage d'utilisateurs ou de serveurs, les risques liés aux nouvelles versions sont considérablement réduits.
- **Test en Environnement Réel** : La nouvelle version est testée dans des conditions réelles, ce qui permet de détecter des problèmes qui pourraient ne pas être apparus dans un environnement de test.
- **Rollback Rapide** : Si des problèmes surviennent, il est facile de revenir rapidement à l'ensemble de réplicas original, garantissant ainsi la stabilité du système.
- **Haute Disponibilité** : En permettant un déploiement contrôlé et progressif des modifications, la stratégie canary assure une disponibilité élevée du système, même en cas de déploiements fréquents.
- **Feedback Précieux** : Les utilisateurs exposés à la nouvelle version peuvent fournir des feedbacks précieux, permettant d'ajuster et d'améliorer la version avant son déploiement global.

2.3 Rollback dans Kubernetes

Le rollback est une fonctionnalité cruciale dans Kubernetes qui permet de revenir à une version précédente d'une application en cas de défaillance ou de comportement inattendu après une mise à jour. Cette capacité de restauration rapide est essentielle pour maintenir la stabilité et la disponibilité des applications dans des environnements de production.

Avantages

Les principaux avantages du rollback dans Kubernetes incluent :

— **Réduction du Temps d'Indisponibilité** : Permet de revenir rapidement à une version stable, minimisant ainsi les interruptions de service.

— **Sécurité et Fiabilité** : Assure une restauration fiable des versions précédentes, augmentant la résilience des applications.

— **Simplicité d'Utilisation** : La commande de rollback est simple à utiliser et s'intègre facilement dans les pipelines CI/CD, facilitant ainsi la gestion des déploiements.

— **Flexibilité** : Permet de tester de nouvelles versions en toute confiance, sachant qu'il est possible de revenir en arrière si nécessaire.

2.4 Conclusion

Dans ce chapitre, nous avons plongé dans l'univers de Kubernetes et ses stratégies de déploiement. Cette immersion dans la gestion des conteneurs jettera les bases nécessaires pour aborder les objectifs spécifiques et la conception détaillée de notre projet dans le chapitre suivant.

Chapitre 3

Spécification des Objectfs et de la Conception

3.1 Introduction

Dans ce chapitre, nous allons présenter les objectifs spécifiques fixés et la conception de notre projet. Nous définirons les différentes étapes clés et les buts à atteindre pour garantir le succès de notre déploiement. En outre, nous décrirons la méthodologie et les principes directeurs qui ont orienté notre approche, afin de poser des bases solides pour la mise en œuvre de notre solution.

3.2 Objectifs

- Créer une pipeline CI/CD automatisée.
- Utiliser Infrastructure as Code (IaC) pour la gestion de l'infrastructure.
- Déployer une application en utilisant différentes stratégies sur Kubernetes.
- Analyser et comparer les stratégies de déploiement en termes de coûts et d'efficacité.
- Mettre en place un système de monitoring efficace.

3.3 Conception

3.3.1 Conception de la pipeline CI/CD

La conception de la pipeline CI/CD (Continuous Integration/Continuous Deployment) vise à automatiser les processus de compilation, de test et de déploiement pour assurer une livraison continue et fiable des nouvelles versions de l'application.

3.3.1.1 Structure de la pipeline

La pipeline CI/CD est structurée en plusieurs étapes :

- **Intégration continue (CI)** :
  - Compilation du code source.
  - Exécution des tests unitaires et d'intégration.
  - Analyse statique du code pour assurer la qualité et le respect des normes de codage.
- **Déploiement continu (CD)** :
  - Construction des images de conteneurs Docker.

— Publication des images sur Azure Container Registry (ACR).
— Déploiement des images sur Azure Kubernetes Service (AKS) en utilisant différentes stratégies de déploiement.

3.3.2 Architecture des déploiements sur Kubernetes

L'architecture des déploiements sur Kubernetes est conçue pour assurer la haute disponibilité, la scalabilité et la résilience des applications.

3.3.2.1 Stratégies de déploiement

Nous utilisons plusieurs stratégies de déploiement pour assurer la continuité du service et minimiser les risques :

— **Rolling Update** : Mise à jour progressive des instances de l'application, remplaçant les anciennes versions par les nouvelles sans interruption majeure.
— **Blue-Green Deployment** : Déploiement de la nouvelle version de l'application dans un environnement parallèle (blue) avant de basculer le trafic du serveur de production (green) vers le nouvel environnement.
— **Canary Release** : Déploiement progressif de la nouvelle version de l'application sur un sous-ensemble de serveurs pour tester et valider les modifications avant un déploiement complet.
— **Recreate** : Suppression de toutes les instances de l'ancienne version avant de créer les nouvelles instances, ce qui entraîne une interruption complète du service pendant le déploiement.

3.3.2.2 Gestion des configurations avec Helm Charts

Helm Charts sont utilisés pour gérer les configurations des applications déployées sur Kubernetes. Un Helm Chart contient tous les fichiers de configuration nécessaires pour déployer une application sur Kubernetes, ce qui simplifie le processus de déploiement et assure la cohérence des configurations.

3.3.3 Utilisation de l'Infrastructure as Code (IaC)

L'Infrastructure as Code (IaC) est une pratique clé pour la gestion automatisée et cohérente de l'infrastructure qui permettera de définir et provisionner les ressources nécessaires.

3.3.3.1 Définition des ressources

Les ressources sont définies dans des fichiers de configuration Terraform, permettant de gérer l'infrastructure de manière déclarative :

— Création des clusters AKS.
— Configuration des registres de conteneurs Azure Container Registry (ACR).
— Gestion des réseaux et des règles de sécurité.

3.3.3.2 Automatisation des déploiements

L'utilisation de Terraform permet d'automatiser le déploiement et la gestion des ressources, facilitant les mises à jour et la gestion des configurations. Les fichiers de configuration sont versionnés et suivis dans un système de contrôle de version, assurant la traçabilité des modifications.

3.3.4 Mise en place du système de monitoring

Un système de monitoring efficace est crucial pour assurer la performance et la disponibilité des applications et pouvoir effectuer la comparaison.

3.3.4.1 Collecte des métriques

Collecter les métriques de performance et de santé des applications déployées sur AKS. Les métriques sont collectées à partir de points d'exposition définis et stockées sous forme de séries temporelles.

3.3.4.2 Visualisation des données colléctées

Visualiser les données collectées. Des tableaux de bord interactifs sont créés pour surveiller en temps réel l'état des applications et détecter rapidement les anomalies.

3.4 Conclusion

Ce chapitre a détaillé les objectifs spécifiques et la conception de notre projet. Nous avons établi des objectifs clairs, comme la création d'une pipeline CI/CD automatisée, l'utilisation de l'Infrastructure as Code (IaC), et le déploiement d'applications sur Kubernetes avec diverses stratégies. Nous avons aussi décrit l'architecture des déploiements, la gestion des configurations, et la mise en place d'un système de monitoring.

Dans le prochain chapitre, nous explorerons les technologies utilisées pour réaliser ces objectifs, notamment Azure Kubernetes Service (AKS), Azure Container Registry (ACR), GitLab, Helm Charts, Terraform, Prometheus, et Grafana.

Chapitre 4

Technologies Utilisées

4.1 Introduction

Dans ce chapitre, nous allons présenter les différentes technologies utilisées dans le cadre de notre projet de déploiement avancé sur Kubernetes. Ces technologies incluent Azure Kubernetes Service (AKS), Azure Container Registry, GitLab pour les pipelines CI/CD, Grafana et Prometheus pour le monitoring, Helm Charts pour la gestion des déploiements, et Terraform pour l'automatisation de l'infrastructure.

4.2 Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) est une solution de gestion Kubernetes entièrement gérée par Microsoft Azure. Elle simplifie le déploiement, la gestion et les opérations de clusters Kubernetes. AKS permet de créer des clusters Kubernetes en quelques minutes, offrant une infrastructure scalable et hautement disponible. Grâce à AKS, nous pouvons déployer nos applications conteneurisées de manière efficace et fiable, tout en bénéficiant de l'intégration étroite avec les autres services Azure.

FIGURE 4.1 – Logo AKS

4.3 Azure Container Registry (ACR)

FIGURE 4.2 – Logo ACR

Azure Container Registry (ACR) est un service de registre de conteneurs Docker privé, entièrement géré et basé sur Azure. Il permet de stocker et gérer les images de conteneurs Docker, facilitant ainsi leur déploiement sur les clusters AKS. En utilisant ACR, nous pouvons sécuriser nos images de conteneurs et les rendre accessibles de manière rapide et fiable lors des déploiements.

4.4 GitLab pour les Pipelines CI/CD

GitLab est une plateforme DevOps complète qui offre des fonctionnalités pour la gestion de code source, les pipelines CI/CD, et bien plus encore. Dans ce projet, nous utilisons GitLab pour créer des pipelines CI/CD automatisés. Ces pipelines permettent de gérer l'intégration et le déploiement continu de nos applications, automatisant ainsi les tests, la construction et le déploiement des conteneurs sur AKS. Grâce à GitLab, nous pouvons assurer une livraison rapide et fiable des nouvelles fonctionnalités tout en minimisant les erreurs humaines.

FIGURE 4.3 – Logo Gitlab

4.5 Grafana et Prometheus

Grafana et Prometheus sont deux outils essentiels pour le monitoring et l'observabilité des applications.

FIGURE 4.4 – Logo Grafana et Prometheus

4.5.1 Prometheus

Prometheus est une solution open-source de surveillance et d'alerte, spécifiquement conçue pour le monitoring des systèmes distribués. Il collecte des métriques depuis des points d'exposition définis et stocke ces données sous forme de séries temporelles. Dans notre projet, Prometheus est utilisé pour collecter les métriques de performance et de santé de nos applications déployées sur AKS.

4.5.2 Grafana

Grafana est une plateforme d'analytique et de monitoring open-source, qui permet de visualiser les données collectées par Prometheus (et d'autres sources) à travers des tableaux de bord interactifs. En utilisant Grafana, nous pouvons créer des visualisations personnalisées des métriques de performance de nos applications, ce qui nous aide à surveiller leur état en temps réel et à réagir rapidement en cas de problème.

4.6 Helm Charts

FIGURE 4.5 – Logo Helm Charts

Helm est un gestionnaire de paquets pour Kubernetes, qui facilite le déploiement et la gestion des applications Kubernetes. Un Helm Chart est un package qui contient tous les fichiers de configuration nécessaires pour déployer une application sur Kubernetes. Dans notre projet, nous utilisons Helm Charts pour automatiser et simplifier le déploiement de nos applications sur AKS. Grâce à Helm, nous pouvons gérer les versions de nos déploiements, rouler facilement des mises à jour et effectuer des rollbacks en cas de besoin.

4.7 Terraform

Terraform est un outil d'infrastructure as code (IaC) open-source développé par HashiCorp, qui permet de définir, provisionner et gérer l'infrastructure à l'aide de fichiers de configuration lisibles par des machines. Dans ce projet, nous utilisons Terraform pour automatiser la création et la gestion de notre infrastructure sur Azure, incluant les clusters AKS, les registres de conteneurs, et d'autres ressources nécessaires. Terraform nous permet de versionner notre infrastructure, de suivre les modifications et de garantir la reproductibilité des environnements, facilitant ainsi le déploiement et la gestion de notre infrastructure de manière cohérente et efficace.

FIGURE 4.6 – Logo Terraform

4.8 Conclusion

En résumé, notre choix d'outils tels qu'Azure Kubernetes Service (AKS), Azure Container Registry, GitLab, Grafana, Prometheus, Helm Charts et Terraform a permis de construire une infrastructure de déploiement robuste. Grâce à ces solutions, nous assurons un déploiement continu, une disponibilité accrue et des performances optimales pour nos applications. Dans le prochain chapitre, nous détaillerons notre plan de mise en œuvre, mettant en pratique ces outils pour concrétiser notre projet.

Chapitre 5

Implémentation et Résultats

5.1 Introduction

5.2 Vue d'ensemble de la Pipeline CI/CD

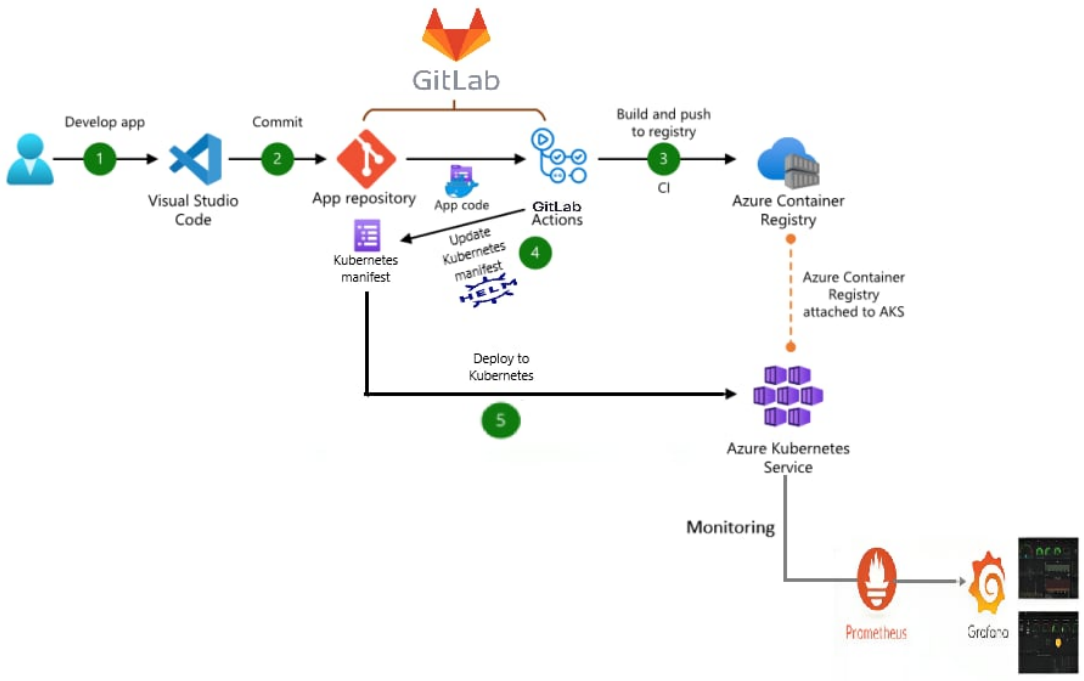

FIGURE 5.1 – Conception de la pipeline CI/CD

L'image ci-dessus présente une vue d'ensemble de la conception de notre pipeline CI/CD, qui guidera la réalisation des différentes étapes du projet. La pipeline se compose des étapes suivantes :

- **Commit de code** : Le code est commité dans le dépôt d'application (App repository) via GitLab.
- **Intégration Continue (CI)** : GitLab Actions est utilisé pour automatiser le processus de construction et de push des images Docker vers Azure Container Registry (ACR).
- **Mise à jour des manifestes Kubernetes** : Les manifestes Kubernetes sont mis à jour et appliqués, déclenchant le déploiement des nouvelles images sur Azure Kubernetes Service (AKS) en utilisant Helm Charts.

— **Déploiement sur Kubernetes et Monitoring** : Les images sont déployées sur AKS, et le processus est monitoré à l'aide de Prometheus et Grafana pour garantir la performance et la disponibilité de l'application.

— **Gestion de l'infrastructure avec Terraform** : Terraform est utilisé pour gérer l'infrastructure de manière déclarative. Les ressources nécessaires, telles que le cluster AKS et le registre ACR, sont définies et provisionnées à l'aide de fichiers de configuration Terraform, ce qui permet d'automatiser et de versionner l'infrastructure, facilitant ainsi les mises à jour et la gestion des configurations.

5.2.1 Préparation de l'environnement

La préparation de l'environnement est une étape cruciale pour assurer le bon déroulement de notre projet. Elle comprend plusieurs sous-étapes essentielles pour établir une base solide avant de passer aux phases de déploiement et de monitoring. Les principales étapes de préparation de l'environnement sont les suivantes :

5.2.1.1 Installation des outils nécessaires

Pour commencer, il est essentiel d'installer tous les outils nécessaires pour le développement et le déploiement de l'application. Les outils incluent, mais ne sont pas limités à :

— **Visual Studio Code** : Un éditeur de code utilisé pour le développement de l'application.

— **Git** : Un système de contrôle de version distribué pour gérer les modifications du code source.

— **Docker** : Un outil permettant de créer et de gérer des conteneurs, utilisé pour empaqueter l'application et ses dépendances.

— **Helm** : Un gestionnaire de paquets pour Kubernetes, utilisé pour déployer, configurer et gérer des applications sur Kubernetes.

— **Terraform** : Un outil d'Infrastructure as Code (IaC) utilisé pour provisionner et gérer l'infrastructure.

— **Azure CLI** : Une interface en ligne de commande pour gérer les ressources Azure, y compris Azure Kubernetes Service (AKS) et Azure Container Registry (ACR).

— **Prometheus et Grafana** : Outils de monitoring et de visualisation des performances et de la santé des applications déployées.

5.2.1.2 Configuration des comptes et des accès

Pour garantir que tous les membres de l'équipe ont accès aux outils et aux ressources nécessaires, les étapes suivantes doivent être effectuées :

— Création de comptes Azure pour accéder aux services tels qu'Azure Kubernetes Service (AKS) et Azure Container Registry (ACR).

— Configuration des accès à GitLab pour le contrôle de version et l'intégration continue.

5.2.1.3 Automatisation des tâches répétitives

Pour améliorer l'efficacité et réduire les erreurs humaines, certaines tâches peuvent être automatisées :

— Configuration de pipelines CI/CD dans GitLab pour automatiser les processus de construction, de test et de déploiement.

— Utilisation de Terraform pour automatiser le provisionnement et la gestion de l'infrastructure.

— Scriptage des étapes de déploiement et de mise à jour des manifestes Kubernetes.

En suivant ces étapes, nous avons préparé un environnement de développement et de déploiement robuste et sécurisé, prêt à accueillir notre application et à supporter les cycles de développement et de déploiement continus.

5.3 Développement de l'application

Dans cette section, nous allons détailler l'application que nous avons travaillé avec tout au long de ce projet. Cette application est composée de deux parties principales :

— **Front-end** : Développé avec Node.js, le front-end fournit une interface utilisateur interactive et réactive.

— **Back-end** : Développé avec Node.js, le back-end gère la logique de l'application.

L'application a été conçue de façon à utiliser des microservices pour une meilleure gestion et une scalabilité accrue, facilitant ainsi le déploiement et la maintenance.

Le choix de cette application a été fait pour modéliser les microservices et pour s'adapter aux limitations du compte Azure, qui permet de déployer des applications de petites tailles.

5.4 Création du cluster AKS

La création du cluster Azure Kubernetes Service (AKS) a été réalisée en utilisant Terraform, un outil d'Infrastructure as Code (IaC). Cette approche permet de définir, provisionner et gérer l'infrastructure de manière déclarative, facilitant l'automatisation et la reproductibilité des configurations. Voici les étapes principales que nous avons suivies :

5.4.1 Configuration de Terraform

Pour commencer, nous avons configuré Terraform pour interagir avec Azure. Cela inclut l'installation de Terraform et la configuration des informations d'identification nécessaires pour accéder à notre compte Azure. Voici les principales étapes :

— Installation de Terraform : Nous avons téléchargé et installé Terraform sur notre machine de développement.

— Configuration des informations d'identification : Nous avons configuré les variables d'environnement Azure, telles que l'ID du client, le secret du client, l'ID de l'abonnement et le tenant.

5.4.2 Définition des fichiers de configuration Terraform

Nous avons ensuite défini les fichiers de configuration Terraform pour créer et configurer le cluster AKS. Les fichiers de configuration principaux incluent :

— `main.tf` : Ce fichier contient les ressources principales nécessaires à la création du cluster AKS, telles que le groupe de ressources, le cluster AKS lui-même et les dépendances.

— `variables.tf` : Ce fichier définit les variables utilisées dans les fichiers de configuration, telles que les noms de ressources, les tailles de machines virtuelles et les configurations de réseau.

5.4.3 Initialisation et application de la configuration Terraform

Une fois les fichiers de configuration définis, nous avons initialisé et appliqué la configuration Terraform pour créer le cluster AKS :

— Initialisation : Nous avons exécuté la commande `terraform init` pour initialiser le répertoire de travail Terraform. Cette commande télécharge les plugins nécessaires et configure l'environnement Terraform.

— Planification : Nous avons exécuté la commande `terraform plan` pour créer un plan d'exécution. Cette étape permet de vérifier les modifications qui seront apportées à l'infrastructure.

— Application : Enfin, nous avons exécuté la commande `terraform apply` pour appliquer la configuration et créer le cluster AKS. Cette commande provisionne les ressources définies dans les fichiers de configuration.

5.4.4 Vérification de la création du cluster

Après l'application de la configuration Terraform, nous avons vérifié la création du cluster AKS :

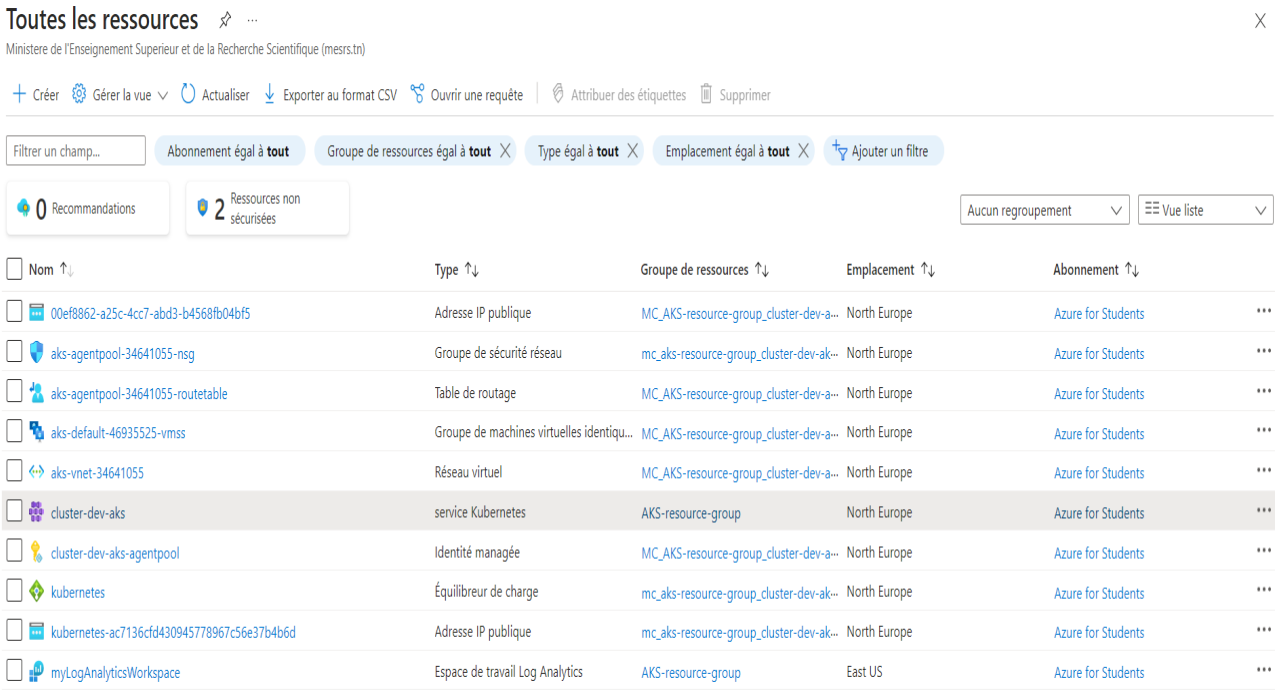

FIGURE 5.2 – Vérification de la création du cluster

Pour se connecter à notre cluster on a utilisé l'outil de ligne de commande pour interagir directement avec le cluster à l'aide de kubectl avec la commande : `az aks get-credentials -resource-group AKS-resource-group -name cluster-dev-aks -overwrite-existing`

En suivant ces étapes, nous avons pu créer et configurer un cluster AKS fonctionnel en utilisant Terraform, facilitant ainsi la gestion de l'infrastructure et la mise en place d'un environnement de déploiement robuste et évolutif.

5.5 Les artéfacts résultants de la pipeline CI/CD

— Images Docker :
  — Image frontend : application/frontend :latest poussée au registre de conteneurs Azure.
  — Image backend : application/backend :latest poussée au registre de conteneurs Azure.
— Journaux (Logs) :
  — Logs de build : Journaux détaillant la construction des images Docker pour le frontend et le backend.
  — Logs de déploiement : Journaux détaillant le processus de déploiement sur AKS via Helm.
— Statut des Pods et Déploiements Kubernetes :
  — Statut des pods avant et après mise à jour : Informations sur les pods avant et après le déploiement.
  — Descriptions des déploiements : Détails des déploiements frontend et backend obtenus via kubectl describe deployment.
— Fichiers Helm Chart :
  — Chart Helm : Fichiers de déploiement Helm utilisés pour déployer les applications sur AKS, y compris les fichiers values.yaml mis à jour.

5.5.1 Configuration de Prometheus et Grafana pour le Monitoring

Pour assurer un monitoring efficace de notre application déployée sur Azure Kubernetes Service (AKS), nous avons mis en place Prometheus et Grafana en utilisant Helm, ce qui nous a permis de tirer parti de leur intégration et de leur automatisation puissantes. Voici les étapes détaillées de la configuration :

5.5.1.1 Installation de Prometheus et Grafana avec Helm

Nous avons choisi d'utiliser Helm pour installer Prometheus et Grafana car cela simplifie et standardise leur déploiement sur notre cluster Kubernetes. Les commandes suivantes ont été utilisées pour l'installation :

— Installation de Prometheus : `helm install prometheus stable/prometheus -namespace monitoring`
— Installation de Grafana : `helm install grafana stable/grafana -namespace monitoring`

Ces commandes déploient Prometheus et Grafana dans le namespace "monitoring", configurant automatiquement les composants nécessaires pour leur fonctionnement optimal.

5.5.1.2 Configuration du Port-forwarding

Pour accéder aux interfaces web de Prometheus et Grafana, nous avons mis en place un port-forwarding local. Cela nous permet de visualiser les dashboards sans exposer notre service de monitoring sur Internet. Les commandes suivantes ont été utilisées :

— Pour Prometheus : `kubectl port-forward service/prometheus-server 9090:9090 -n monitoring`
— Pour Grafana : `kubectl port-forward service/grafana 3000:3000 -n monitoring`

Cela redirige les ports locaux vers les services correspondants dans le cluster, permettant un accès sécurisé via 'localhost'.

5.5.1.3 Création d'un Service Principal et Attribution des Rôles

Pour permettre à Grafana d'accéder aux métriques d'Azure Monitor pour les conteneurs, nous avons créé un Service Principal dans Azure. Ce Service Principal a été configuré avec les rôles nécessaires pour accéder aux ressources du groupe de ressources AKS. Cela inclut la lecture des métriques et des logs, essentielle pour un monitoring détaillé.

5.5.1.4 Configuration de la Source de Données dans Grafana

Une fois Grafana opérationnel, nous avons configuré une source de données qui pointe vers notre instance Prometheus. Cela permet à Grafana de récupérer et de visualiser les métriques collectées par Prometheus. La source de données a été ajoutée via l'interface utilisateur de Grafana, en spécifiant l'URL de Prometheus comme endpoint.

5.5.1.5 Importation d'Azure Monitor pour les Conteneurs dans Grafana

Nous avons ensuite importé le plugin "Azure Monitor for Containers" dans Grafana, ce qui nous permet de visualiser non seulement les métriques de Prometheus mais aussi celles provenant directement d'Azure Monitor. Ce plugin enrichit notre tableau de bord avec des métriques spécifiques à Azure, offrant une vue plus complète de notre environnement.

5.5.1.6 Visualisation des Métriques sur le Tableau de Bord de Grafana

Enfin, avec toutes les configurations en place, nous avons créé et personnalisé des tableaux de bord dans Grafana pour visualiser les métriques clés de notre application et de l'infrastructure sous-jacente. Ces tableaux de bord nous permettent de surveiller la performance, la disponibilité et la santé de notre application en temps réel.

Ces étapes ont permis de mettre en place une solution de monitoring robuste et intégrée, essentielle pour maintenir la performance et la stabilité de notre application déployée sur AKS.

5.6 Mise en Place des Stratégies de Déploiement

5.6.1 Stratégie Recreate :

Dans cette section, nous allons détailler la démarche que nous avons choisie pour déployer et gérer notre application en utilisant la stratégie Recreate. Tout d'abord, nous avons créé un Helm chart spécifique pour déployer notre application. Ce chart inclut toutes les configurations nécessaires pour déployer les différents composants de notre application de manière orchestrée. Pour chaque modification de code, nous avons mis en place une pipeline CI/CD en utilisant GitLab. Cette pipeline est conçue pour automatiser le processus de mise à jour du chart Helm. Chaque fois qu'une nouvelle modification est poussée (push) dans le dépôt, la pipeline se déclenche, compile et teste le code, puis met à jour le chart Helm en utilisant la méthode Recreate. Cette méthode permet de déployer les nouvelles versions de nos pods en supprimant les anciens pods avant de créer les nouveaux. Cela garantit que nous avons toujours la version la plus récente de notre application en cours d'exécution sans laisser d'anciennes versions résiduelles.

La manipulation spécifique de la stratégie de déploiement s'est faite à travers les fichiers de manifestes, précisément dans le fichier `deployment.yaml`, en introduisant `strategy: type: Recreate`, ce qui force Kubernetes à supprimer tous les pods existants avant d'en lancer de nouveaux à chaque mise à jour.

En utilisant le système de monitoring mis en place, on a réussi à visualiser en temps réel le comportement des pods et des déploiements depuis notre propre machine. Grâce à cette configuration, nous pouvons observer les variations de disponibilité des réplicas et d'autres métriques essentielles lorsque nous utilisons la stratégie de recréation.

5.6.1.1 Résultats

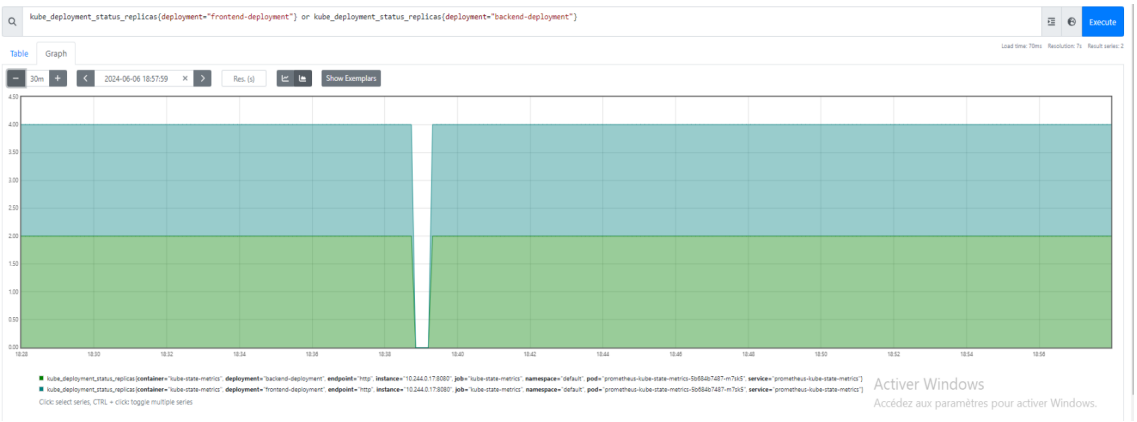

FIGURE 5.3 – Statut du déploiement après une mise à jour Recreate

La surveillance des deux Déploiemnts FrontEnd et BackEnd , nous a permis d'observer un temps de downtime lors de la transition entre les versions. En effet, la suppression de tous les pods suivie de la recréation des pods avec la nouvelle version a entraîné un downtime de 2 secondes.

Ce downtime, bien que présent, reste relativement court, car notre application est minimaliste avec peu de dépendances. Cependant, pour une application ayant une taille de dépendances élevée, ce downtime pourrait être significativement plus long, impactant ainsi la disponibilité et l'expérience utilisateur.

5.6.1.2 Conclusion

La stratégie Recreate présente donc des avantages et des inconvénients. Elle garantit que toutes les instances de l'application sont mises à jour simultanément, mais elle engendre un temps de downtime inévitable. Pour des applications plus complexes ou critiques, il serait pertinent d'explorer d'autres stratégies de déploiement,

5.6.2 Stratégie Rolling Update

La stratégie Rolling Update est la méthode par défaut utilisée pour le déploiement des applications dans Kubernetes via Helm. Cette stratégie permet de mettre à jour les instances d'une application de manière séquentielle sans interruption du service, ce qui la rend idéale pour les environnements de production où la disponibilité continue est critique.

Comme pour la stratégie Recreate, nous avons utilisé un Helm chart spécifique pour gérer les déploiements de notre application.

La grande différence avec Recreate est que, au lieu de supprimer tous les pods avant de déployer les nouveaux, Kubernetes met à jour les pods un par un, en s'assurant qu'il y a toujours des instances de l'application en cours d'exécution.

Dans cette stratégie, nous avons utilisé deux paramètres importants : `maxSurge` et `maxUnavailable`. Le paramètre `maxSurge` définit le nombre de pods qui peuvent être créés au-delà du nombre souhaité de pods pendant une mise à jour. Cela peut être un nombre absolu ou un pourcentage du nombre de réplicas, avec une valeur par défaut de 25 pourcent.

Le paramètre `maxUnavailable` définit le nombre de pods qui peuvent être indisponibles pendant le processus de mise à jour. Cela peut également être un nombre absolu ou un pourcentage du nombre de réplicas, avec une valeur par défaut de 25 pourcent .

Dans la mise en place de cette stratégie, nous avons laissé ces paramètres par défaut.

5.6.2.1 Résultats

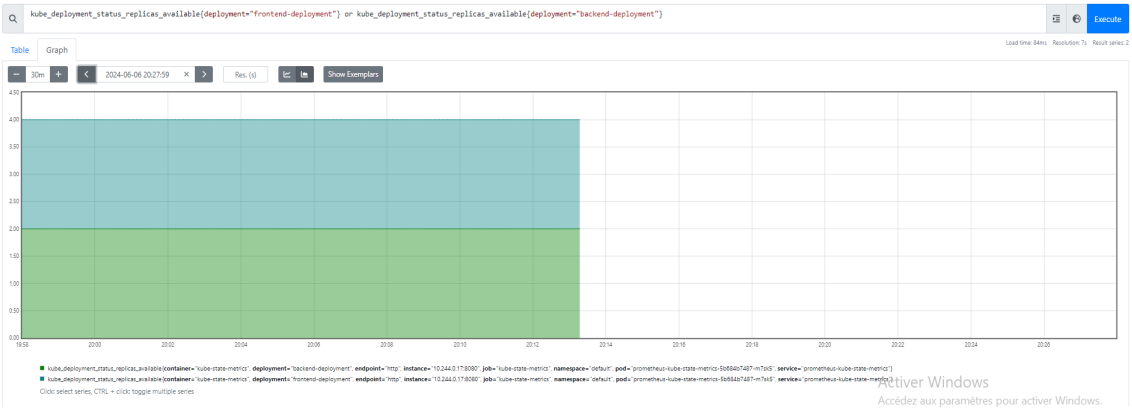

FIGURE 5.4 – Statut du déploiement après une mise à jour Rolling Update

La surveillance des deux Déploiemnts FrontEnd et BackEnd , nous a permis d'observer un temps de downtime nul lors de la transition entre les versions.

5.6.3 Conclusion

La stratégie Rolling Update offre plusieurs avantages significatifs : Zéro Downtime : Les utilisateurs de l'application ne subissent aucun arrêt du service car il y a toujours des pods fonctionnels pendant la mise à jour. Contrôle et Flexibilité : Il est possible de contrôler la vitesse de déploiement et le nombre de pods mis à jour simultanément, grâce aux paramètres de configuration comme le nombre maximal de pods indisponibles et le nombre maximal de nouveaux pods créés.

En conclusion, la stratégie Rolling Update permet une gestion efficace et sûre des déploiements continus de notre application. Elle assure une haute disponibilité et minimise les risques lors des mises à jour, ce qui est crucial pour le maintien de services critiques en ligne.

5.6.4 Stratégie Blue-Green

5.6.5 Stratégie Canary

5.6.6 Résultats et Comparaison

Chapitre 6

Conclusion Générale

Synthèse des résultats obtenus, des défis rencontrés, et des leçons apprises. Comparaison des différentes stratégies de déploiement et recommandations pour les entreprises.

Bibliographie

[1] Kubernetes https://kubernetes.io/docs/home/

[2] Terraform https://www.terraform.io/

[3] Helm Charts https://helm.sh/docs/topics/charts/

[4] AKS https://learn.microsoft.com/en-us/azure/aks/