Dédicaces

A mes parents, pour vos sacrifices, vos conseils et vos bénédictions.

A ma soeur, pour le soutien que vous m'avez accordé,

A mes amis, pour les moments qu'ils m'ont faits vivre, pour la joie avec laquelle ils ont parsemé ma vie.

A tous ceux qui ont participé à ma formation,

A tous ceux qui m'ont aidé à réaliser ce travail

Remerciement

Avant tout développement sur le déroulement de mon stage PFE, il paraît opportun de commencer ce rapport par des remerciements à ceux qui ont participé de différentes façons à la réussite de ce stage et plus particulièrement les personnes que je cite ci-dessous.

De ce fait, je tiens, dans un premier temps, à exprimer toute ma gratitude à l'égard de Monsieur Abderrazak Hachani, mon encadrant, pour sa disponibilité et son accompagnement tout au long de mon projet.

Je tiens également à remercier Monsieur Raphael Monrouzeau l'expert DevOps et Monsieur Julien Beaucourt le manager technique, pour leur accueil et ses conseils. Mon stage de la société Mikado Labs a été très formateur et m'a surtout permis de m'immerger dans un secteur qui m'intéressait depuis longtemps et m'a permis également de consolider mes connaissances dans le domaine du Cloud Computing, ma discipline qui me fascine au plus haut point.

A tout le staff de la société Mikado Labs, pour la disponibilité la sympathie et la gentillesse dont ils ont fait preuve durant la période de mon stage.

Par ailleurs, je suis particulièrement reconnaissant à tous les enseignants d'ESPRIT qui, grâce à leur pédagogie exemplaire, m'ont transmis les connaissances théoriques nécessaires pour l'accomplissement de ce projet.

Je tiens aussi à exprimer ma profonde gratitude aux membres du jury de m'avoir fait l'honneur de participer à l'évaluation de mon travail.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

API : Application programming interface

CI/CD : Continious Integration/ Continious deployment

DNS : Domain Name Service

HTTP : HyperText Transfer Protocol

JSON : JavaScript Object Notation

GCP : Google Cloud Platform

GKE : Google Kubrernetes Engine

NFS : Network file system

PVC : Persistant voluem claim

PV : Persistant volume

UML : Unified Modeling language

Introduction Générale

Aujourd'hui, face au boom et l'explosion de la data, la gestion de ces données et de l'ensemble des systèmes devient de plus en plus complexe.

Face à cette difficulté les entreprises cherchent à externaliser et confier leurs problèmes à des prestataires. C'est le cas, par exemple, avec le développement du Cloud Computing qui retient et suscite l'intérêt des entreprises qui souhaitent migrer vers ce modèle innovant.

En effet, Le cloud a posé une véritable révolution concernant le mode d'approvisionnement et d'exploitation des ressources IT, par ce biais là les sociétés limitent l'investissement dans la gestion, en ce qui concerne les ressources informatiques qui sont lourdes et très couteuses, et en ce qui concerne le cycle de vie des projets.

Reconnaissant le stress que cela impose aux équipes informatiques, une nouvelle philosophie est née, réconciliant les développeurs et les opérationnels, sous le nom de « DevOps ». Des outils spécifiques ont été inventés pour automatiser les tâches hautement techniques et sujettes aux erreurs, pourtant monotones, du développement et du déploiement de logiciels.

En outre, plusieurs sociétés se sont spécialisées dans le domaine de Cloud Computing ont adopté l'approche « DevOps » afin d'accompagner et d'aider les différentes entreprises dans leurs problématiques de gestion de leur système d'information (transformation digitale).

Mikado Labs suit cette dynamique dans laquelle s'inscrit mon projet de fin d'études. En s'inspirant du mouvement 'DevOps', nous avons mis en place une plateforme de service en conteneurs 'CaaS' basée sur une architecture Cloud afin de migrer une multitude de sites web vers cette plateforme dans un pipeline CI/CD.

On présentera dans ce qui suit l'organisation générale du présent rapport qui s'articule autour de quatre chapitres. Nous présentons, d'abord, un premier chapitre, intitulé "Cadre du projet" où nous présentons l'organisme d'accueil, nous exposons notre étude de l'existant et constatations, nous enchaînons avec l'objectif du projet et notre contribution, pour finir avec la méthodologie de travail adoptée. Notre deuxième chapitre intitulé "Étude et choix des outils techniques" permet de clarifier les concepts de base dans notre solution ainsi que la comparaison entre les technologies les plus utilisés sur le marché et susceptibles de répondre à notre besoin ensuite nous procédons à la synthèse de nos choix. Au cours de notre troisième

chapitre qui est "Conception" nous identifions les acteurs de notre système, nous spécifions les besoins fonctionnels, techniques et non fonctionnels décelés. Puis, nous procédons à la représentation de ces besoins.

Le chapitre quatre "Réalisation" du système mis en place, permet de présenter l'environnement matériel et logiciel que nous avons utilisé pour aboutir à notre travail. Et pour finir, d'exposer quelques interfaces graphiques relatives à notre travail. Nous clôturons ce rapport par une conclusion générale dans laquelle nous évaluons les résultats atteints et nous exposons les éventuelles perspectives du présent projet.

Chapitre I . Cadre du projet

Plan

Introduction

Ce premier chapitre est consacré à la description du contexte général du projet.

Tout d'abord, nous présenterons l'organisme d'accueil Mikado Labs et l'environnement professionnel dans lequel nous avons pu progresser durant la période de stage. Ensuite, nous décrirons le contexte du projet, en commençant par analyser et critiquer la solution existante, étaler la problématique, et finir par présenter la solution proposée. Enfin, nous expliquerons le choix de la méthodologie de travail qui nous a permis de réaliser ce projet.

I.1. Présentation de l'organisme d'accueil : Mikado Labs

Depuis sa création en 2016, Mikado Labs accompagne ses clients dans le conseil technologique et la transformation de leurs systèmes informatiques à travers une expertise sans faille, cette société a su tirer profit de ses qualités et de sa vigueur pour remporter des grands projets sur le marché.

Aujourd'hui, Mikado Labs est présent sur des grands comptes, notamment des sociétés du CAC 40 (Bourse de Paris) et offre principalement une expertise qui se traduit par des outils et des méthodes modernes pour mettre en place l'infrastructure avec des technologies de pointe, afin de rendre fiable la livraison des multitudes services et produits.

Les cœurs de métiers de Mikado Labs sont les suivants :

- **Cloud / Devops** : Mikado Labs apporte du conseil technologique autour du Cloud Computing, qu'il soit privé ou public, accompagne ses clients sur leurs problématiques et leurs projets de transformation.
- **Transformation IT** : Mikado Labs intervient en conseil sur les systèmes d'informations pour les rendre toujours plus agiles face aux ambitions et l'évolution des métiers.
- **Excellence IT** : Mikado Labs a développé une expertise forte sur la plateforme ServiceNow et apporte du conseil technologique et fonctionnel dans la transformation digitale.
- **Service Manager** : Mikado Labs assure des missions de production informatique réseaux, systèmes et sécurité depuis son centre de services [1].

La figure I.1 illustre ces diverses technologies.

Figure I.1. Les domaines d'interventions de la société Mikado Labs

I.2. Contexte du projet

Mon stage est effectué au sein des locaux de Mikado Labs situés à Boulogne-Billancourt (92100, France) en banlieue Parisienne, pour Cosmo, client de Mikado Labs, sur tous leurs besoins DevOps.

Le projet est sous la supervision du CTO Julien Beaucourt ainsi que l'expert DevOps Raphael Monrouzeau. Ils m'ont aidé dans un premier lieu à s'intégrer dans la société et dans l'équipe de Mikado Labs, dans un second lieu, ils m'ont fait monter en compétence sur divers sujets afin de mettre en place le projet dans les meilleures conditions.

L'objectif du stage consiste à la livraison automatique d'une plateforme de livraison continue basée sur une architecture de conteneurisation gérée par Google.

Nous présenterons dans ce qui suit l'étude et la problématique de la solution existante. Nous enchainerons par la suite, par la description de la solution proposée.

I.2.1. Etude de l'existant

Actuellement notre client souffre de plusieurs problèmes qui peuvent altérer la stabilité des services et la performance de l'infrastructure actuelles.

Ce problème est lié directement au manque de la préoccupation des développeurs de l'impact de leurs codes sur la production. En revanche, ils sont concentrés sur la livraison fréquente de nouvelles fonctionnalités. Chaque développeur installe son serveur web avec sa configuration habituelle et sa base de données sur son système d'exploitation (Windows/Ubuntu/ Redhat...) sans penser aux problèmes de compatibilité qui peuvent survenir sur le serveur de recette ou de production.

Autre que le problème de performance au niveau de l'environnement de déploiement, la configuration des serveurs pré-production et production est faite via des tâches manuelles et répétitives de même pour les tests fonctionnels des applications.Ceci prend une charge majeure et coûteuse en termes de temps et systématiquement en termes de « Time To Market ».

I.2.2. Problématique

Après un audit effectué par notre expert DevOps de Mikado Labs chez le client, nous avons détecté les problèmes auxquels il fait face :

- Déploiement de l'ensemble des services dans un centre de données local (data center) qui regroupe des serveurs, des sous-systèmes de stockage et réseaux, ce qui a pour effet un coût élevé pour l'entreprise cliente.
- Problèmes au niveau de livraison des applications vers l'environnement de production.
- Certains incidents du réseau d'onduleurs ont causé des pertes des services.
- Difficultés d'orchestration des services et des conteneurs Docker.
- Mise à l'échelle manuelle de l'infrastructure.

I.2.3. Solution proposée

Après analyse des différentes problématiques, et d'après les résultats de l'audit, nous avons rédigé les solutions proposées afin de répondre aux différents problèmes que le client rencontre. Une transformation digitale est donc nécessaire par le biais de la méthode AGILE

dans un approche « DevOps » au travers de différentes technologies pour la mise en place de l'infrastructure cloud.

Notre solution consiste à la fourniture automatique d'une plate-forme de conteneurisation gérer par un fournisseur de Cloud afin d'offrir une infrastructure moderne de conteneurs à notre client. Ce type de plate-forme permet à notre client de bénéficier de l'expertise d'autrui en matière d'intégration des technologies de conteneurs. Par exemple, un moteur de conteneurs à lui seul n'est pas suffisant pour le développement d'un produit (à base de conteneurs) prêt pour la production. Cela passe aussi par des aspects tels que l'orchestration, l'automatisation, la capacité de montée en charge et la sécurité. En utilisant cette plate-forme, notre client peut se consacrer en priorité à la réalisation des objectifs métiers au lieu d'avoir à faire face aux complexités de la création d'une infrastructure de conteneurs.

A cet effet, nous serons en mesure de :

- Implémenter une solution d'orchestration de conteneurs qui répond aux exigences de l'entreprise.
- Assurer la haute disponibilité et la mise en échelle automatique de l'infrastructure.
- Elaborer une solution de reprise après sinistre.
- Implémenter une solution d'intégration continue évolutive.
- Implémenter une solution d'orchestration de volumes dans le cloud qui assure la distribution et la réplication des données afin de minimiser les pertes des données.
- Faciliter la gestion et le déploiement des applications conteneurisé
- Automatiser toutes les taches de mise en place de notre solution

La finalité de notre solution est de fournir à notre client les capacités requises pour atteindre ses objectifs avec une rapidité sans précédent. Ce type d'approche leur permet par ailleurs de réaliser des économies dans le développement de solutions et la maintenance de leur infrastructure.

I.3. Méthodologie de travail

Afin que la mise en œuvre du projet se déroule adéquatement, il est nécessaire de suivre une méthodologie de travail pour situer les différentes phases du projet dans le temps et suivre son évolution. C'est la présente partie qui décrit la méthodologie adoptée.

I.3.1. Tableau comparatif

Modèle en cascade	Modèle en spriale
Le modèle en cascade fonctionne de manière séquentielle.	Itératif et incrémental, fonctionne dans la méthode évolutive.
En cascade, les erreurs de modèle ou les risques sont identifiés et corrigés une fois les étapes terminées.	Dans le modèle en spirale, les erreurs ou les risques sont identifiés et corrigés plus tôt
Le modèle en cascade est applicable pour les petits projets.	Le modèle Spiral est utilisé pour les grands projets.
Résistance voire opposition au changement.	Accueil favorable au changement inéluctable, intégré dans le processus.
Processus distinct rigoureux de gestion des risques.	Gestion des risques intégrés dans le processus global, avec responsabilisation de chacun dans l'identification et la résolution des risques.

Tableu I.1. Tableau comparatif des méthodes de travail

Synthèse :

Nous avons mis la choix de le modèle en spirale, car il est plus utilisées pour les gros, de plus, il offre une meilleure adaptabilité, visibilité et gestion des risques. il pourra tout aussi bien être utilisé pour les projets où il n'y pas de documentations détaillées, le client peut alors voir l'évolution du projet et l'adapter selon ses besoins. En revanche, les méthodes classiques seront plus utilisées s'il y a une idée très précise du projet avec un cahier des charges et planning très détaillé où tous les risques possibles sont anticipés.

I.3.2. Modèle de processus en Spirale

Pour la réalisation de notre projet nous avons opté pour une approche caractérisée par un style de conduite de projets itératif, incrémental et adaptatif. Son action est centrée sur l'autonomie de ressources humaines impliquées dans la spécification et la validation des différents modules intégrés et testés en continu. C'est le modèle en spirale. Dans le cadre de ce type de projet, chaque acteur élabore sa vision du produit à réaliser et spécifie les différentes fonctionnalités et exigences. Pour cette raison, le projet doit être réparti en incréments afin de livrer un module après chaque incrément. Le modèle en spirale met l'accent sur l'activité d'analyse des risques et chaque cycle de la spirale se déroule en quatre phases illustrées dans la Figure I.2 [2].

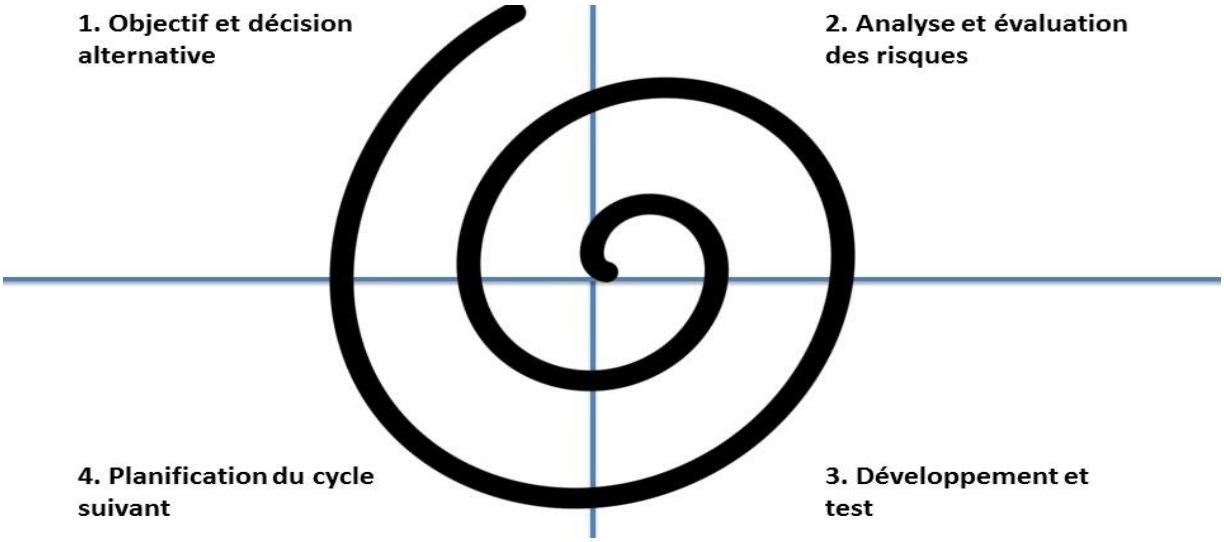

Figure I.2. Les phases de déroulement du cycle en spirale

Le modèle en spirale se caractérise par les cycles suivants :

- **Objectif et décision alternative** : les objectifs sont déterminés conjointement avec le client. Dans le même temps, les alternatives possibles sont discutées et les conditions cadres sont spécifiées (par exemple le système d'exploitation, l'environnement et langage de programmation).
- **Analyse et évaluation des risques :** les risques potentiels sont identifiés et évalués. Les alternatives en question sont également évaluées, tandis que les risques sont enregistrés, estimés puis réduits à l'aide de prototypes, des simulations et des

logiciels d'analyse.Dans ce cycle, plusieurs prototypes existent sous forme de modèles de conception ou de composants fonctionnels.

- **Développement et test :** les prototypes sont encore plus étendus et des fonctionnalités sont ajoutées. Le code réel est écrit, testé et migré vers un environnement de test plusieurs fois jusqu'à ce que le logiciel puisse être implémenté dans un environnement productif.
- **Planification du cycle suivant :** le cycle à venir est planifié à la fin de chaque cycle. Si des erreurs se produisent, les solutions sont recherchées. Si une meilleure alternative est une solution envisageable, elle sera préférée au sein du cycle suivant [2].

1.3.3. Chronogramme des tâches

Dans cette partie, nous définissons la répartition des tâches du projet en fonction du temps tout au long de ces cinq mois de stage au sein de la société Mikado Labs. Cette répartition sera illustrée à travers un diagramme de Gantt décrivant le déroulement de notre projet représenté par la figure ci-dessous.

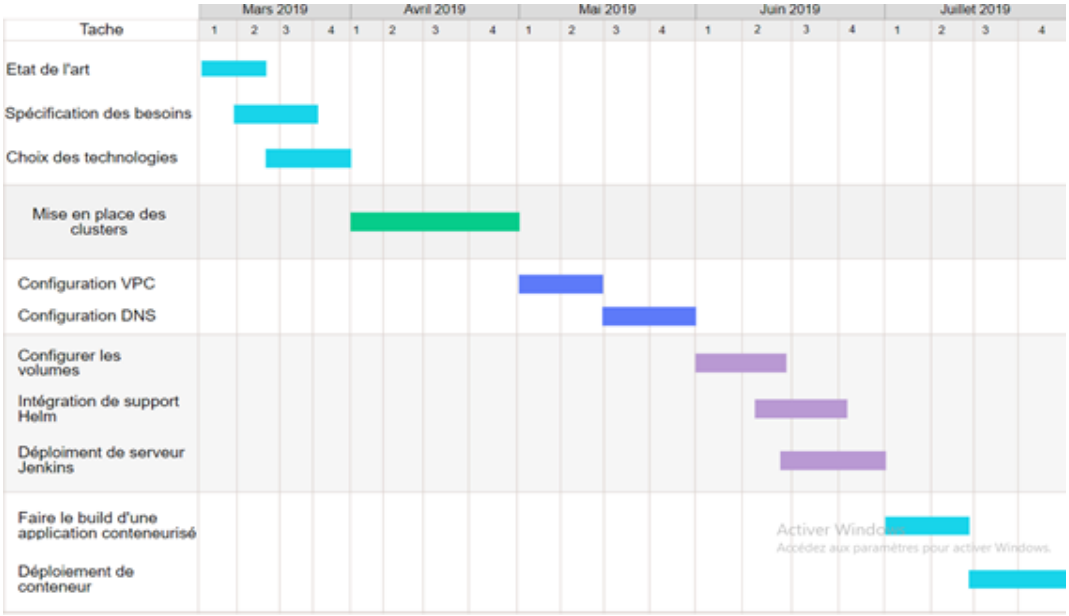

Figure I.3. Diagramme de Gantt

I.4. L'approche Devops

L'approche DevOps est une politique cruciale adoptée par les entreprises les plus modernes permettant de répondre plus rapidement aux feedbacks de leurs clients et d'améliorer la gouvernance.

C'est exactement le processus adopté par toutes les grandes entreprises pour développer des logiciels de haute qualité et des cycles de vie de développement plus courts, ce qui entraîne une plus grande satisfaction de la clientèle, ce que chaque entreprise souhaite.

Figure II. 6. Cycle de chaine DevOps

C'est un cycle sans fin et le logo de DevOps se semble parfaitement logique. Il suffit de regarder la figure ci-dessus.

DevOps est une approche de développement logiciel qui implique un développement continu, des tests continus, une intégration continue, un déploiement continu et une surveillance continue du logiciel tout au long de son cycle de développement [3].

Nous présenterons par la suite, les avantages et l'effet de l'approche DevOps sur la politique de l'entreprise.

I.4.1. Avantages du DevOps :

Avantages techniques:

- Livraison logicielle continue
- Problèmes moins complexes à résoudre
- Résolution plus rapide des problèmes

- Amélioration de la qualité et de la fiabilité du service.

Avantages commerciaux:

- Livraison plus rapide des fonctionnalités
- Des environnements d'exploitation plus stables
- Plus de temps disponible pour ajouter de la valeur (plutôt que de réparer / maintenir)
- Adaptation plus rapide aux changements du marché.

I.4.2. Alignement des équipes

L'approche DevOps joue un rôle essentiel dans le bon déroulement du projet, grâce à une multitude d'outils qui assurent la bonne communication entre les équipes de développements et les équipes relationnels, qui doivent être résilientes en cas de changement du code ou d'architecture.

Voyons maintenant comment DevOps prend en charge les défis rencontrés par le développement et les opérations. Le tableau ci-dessous décrit comment DevOps aborde les défis de développement.

	Dev défis	DevOps solutions
	Temps d'attente pour le déploiement du code	**-Intégration continue** assure un déploiement rapide du code, des tests plus rapides ainsi qu'un mécanisme de feedback **-Livraison continue** Il n'y a pas de temps d'attente pour déployer le code, donc le développeur se concentre sur la construction du code actuel
	Pression du travail sur l'ancien et le nouveau code, en raison du temps requis pour le développement et le déploiement.	

Tableau II.2. Dev défis et Devops solutions

Pour aller plus loin, le tableau ci-dessous décrit comment DevOps répond aux défis opérationnels.

	Ops défis	DevOps solutions
	Difficile de maintenir une disponibilité d'environ 100% de l'environnement de production.	**Conteneurisation / Virtualisation** offres un environnement simulé pour exécuter le logiciel car le conteneur offre une grande fiabilité pour la disponibilité du service

	Les outils d'automatisation d'infrastructure ne sont pas très affectifs.	**Infrastructure en tant que code** Des outils et des systèmes sont en place pour une administration facile.
	Le nombre des serveurs à surveiller continue à augmenter avec le temps et donc la complexité.	-**Surveillance continue** pour fournir un logiciel de qualité au client à un rythme très rapide.
	Il était très difficile de faire des diagnostics et de fournir des feedbacks sur les produits.	-Un système efficace de suivi et de retour d'information devrait être en place.

Tableau II.3. Ops défis et DevOps solutions

I.5. Conteneurisation

I.5.1. Environnements virtuels

La virtualisation est une technologie logicielle éprouvée offrant la possibilité d'exécuter simultanément plusieurs systèmes d'exploitation et applications sur un même serveur. Celle-ci bouleverse rapidement le paysage informatique en modifiant fondamentalement nos usages de la technologie. La virtualisation est un mécanisme informatique qui permet d'exécuter plusieurs systèmes d'exploitation, serveurs ou applications sur une seule machine physique.

C'est un ensemble de techniques qui peuvent être matérielles ou logicielles et qui consistent à réaliser une abstraction des caractéristiques physiques de ressources informatiques, afin de les présenter à des systèmes, des applications ou des utilisateurs.

Il s'agit donc de diviser une ressource physique (serveur, périphérique de stockage...) en plusieurs ressources logiques et agréger plusieurs ressources physiques (périphériques de stockages, serveurs) en une ressource logique. La figure suivante illustre la différence entre une architecture traditionnelle et une architecture virtualisée et les composants de chacune.

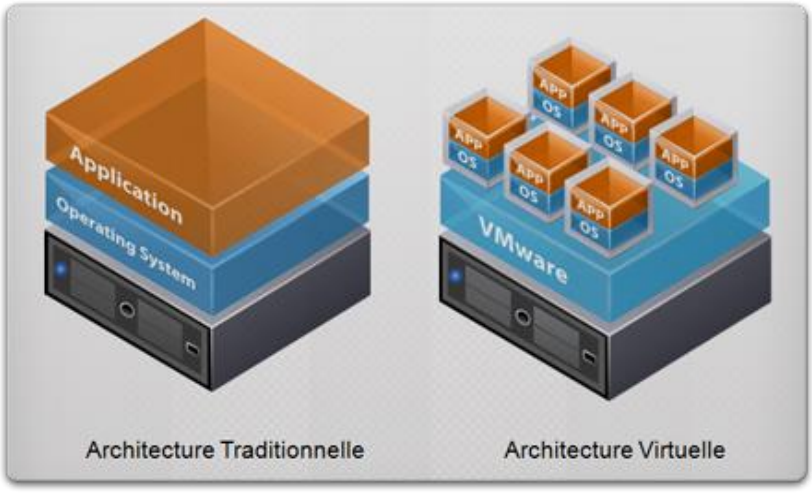

Figure II.9. Comparaison entre une architecture traditionnelle et virtualisée

I.5.2. Conteneurs

La virtualisation à base de conteneurs est une méthode de virtualisation dans laquelle la couche de virtualisation s'exécute au sein même du système d'exploitation (OS).

Dans cette approche, le noyau de l'OS hôte exécute directement plusieurs machines virtuelles (VM) invitées et autonomes sans passer par une couche logicielle supplémentaire. Ces machines invitées s'appellent des conteneurs.

Les conteneurs permettent donc de s'affranchir des hyperviseurs et de la gestion qui découle de l'exécution, sur chaque VM, d'un système d'exploitation invité [4]

La figure II.10 présente la différence entre l'architecture de la mchine virtuelle et deconteneurs.

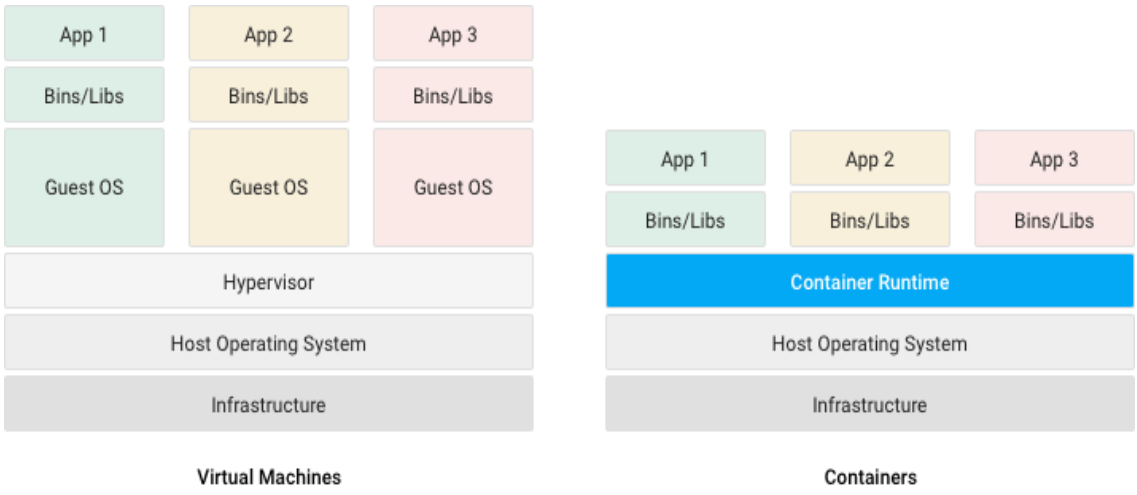

Figure II.10. Machines virtuelles et Conteneurs

Les conteneurs proposent un mécanisme de regroupement logique qui permet d'extraire des applications de l'environnement dans lequel elles s'exécutent réellement. Une fois extraites, les applications basées sur des conteneurs peuvent facilement être déployées dans n'importe quel environnement, qu'il s'agisse d'un centre de données privé ou du cloud public.

Un conteneur consiste en un environnement d'exécution complet qui contient une application et toutes ses dépendances, ses bibliothèques et autres fichiers binaires, ainsi que les fichiers de configuration nécessaires pour l'exécuter, regroupés dans un seul package. En conteneurisant la plate-forme d'application et ses dépendances, les différences dans les distributions du système d'exploitation et l'infrastructure sous-jacente sont abstraites.

I.6. Orchestration

Dans le développement moderne, les applications ne sont plus monolithiques, au contraire elles sont composées de dizaines voire de centaines de composants mis en conteneurs, peu structurés, qui doivent fonctionner ensemble pour permettre à telle ou telle application de fonctionner correctement. L'orchestration de conteneur désigne le processus d'organisation du travail des composants individuels et des niveaux d'application.

Les moteurs d'orchestration de conteneurs permettent aux utilisateurs de contrôler le moment où les conteneurs commencent et s'arrêtent, les grouper en clusters et coordonner tous les processus qui composent une application. Les outils d'orchestration de conteneur permettent aux utilisateurs de guider le déploiement de conteneur et d'automatiser les mises à jour, le contrôle d'état et les procédures de basculement [5].

I.7. Automatisation

L'automatisation de l'infrastructure, consiste à utiliser des logiciels pour créer des instructions et des processus reproductibles dans le but de remplacer ou de réduire l'interaction humaine avec les systèmes informatiques. Les logiciels d'automatisation s'exécutent dans les limites de ces instructions, outils ou structures afin de réaliser des tâches avec une intervention humaine minimale, voire nulle.

L'automatisation est un élément clé de l'optimisation de l'environnement informatique et de la transformation numérique. Les environnements informatiques dynamiques et modernes doivent pouvoir évoluer plus rapidement que jamais, et l'automatisation informatique joue là un rôle essentiel [6].

Conclusion

Ce premier chapitre nous a permis de définir le cadre général de notre projet, à savoir l'entreprise d'accueil, la problématique du projet, l'étude de l'existant, la méthodologie de travail ainsi que le déroulement du stage. Nous allons entamer dans le prochain chapitre la description des aspects fondamentaux du projet afin d'épurer les notions importantes à sa compréhension.

Chapitre II. Étude et choix des outils techniques

Plan

Introduction

Après avoir déterminé les besoins fonctionnels et non fonctionnels de notre projet, nous commencerons dans ce chapitre par réaliser des études comparatives des solutions techniques existantes. Nous définirons après les solutions choisies et nous finirons par présenter l'architecture globale de notre solution.

II.1. Critères de choix des technologies

Dans cette phase, nous avons étudié les différentes possibilités qui s'offrent à nous et nous avons abouti aux conclusions suivantes :

- ✓ Le meilleur choix d'une technologie ne repose pas uniquement sur son efficacité.
- ✓ Nous choisissons des technologies matures et pertinentes plutôt que les technologies instables et qui risquent leur pérennité à l'avenir.
- ✓ Nous utilisons des outils Open-source car nous sommes convaincus par la cause.
- ✓ Nous sommes toujours en avance sur les dernières technologies mais un travail de diffusion des connaissances est primordial pour que toute l'équipe soit aux normes et que les compétences soient au maximum homogènes.

II.2. Solution de conteneurisation :

Pour la solution de conteneurisation, nous avons été amenés à travailler avec Docker pour plusieurs raisons :

- Docker est déjà implémenté dans la plupart des applications hébergées chez le client et déployées sous forme de conteneurs Docker et se base sur des fichiers Docker compose.
- Cette technologie est déjà maîtrisée par les membres de l'équipe.
- Il assure une rapidité et facilité de déploiement des applications.

Docker présente une plateforme logicielle open source permettant de créer, de déployer et de gérer des conteneurs d'applications virtualisées sur un système d'exploitation.

Les services ou fonctions de l'application et ses différentes bibliothèques, fichiers de configuration, dépendances et autres composants sont regroupés au sein du conteneur. Chaque conteneur exécuté partage les services du système d'exploitation.

Cet outil permet en effet de tourner une application sur n'importe quel environnement de la même manière vu qu'il est agnostique par rapport au système. La technologie Docker encourage le travail collaboratif entre développeurs et administrateurs et fournit un environnement qui réduit les silos entre les différentes équipes. L'orchestration est l'une des technologies qui orbite autour de Docker, Google a créé Kubernetes, Amazon a créé ECS, Mesos a lancé Marathon et Docker Inc. a lancé l'initiative Swarm [7].

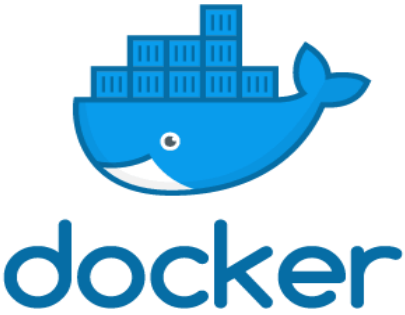

Figure II.7. Logo de Docker

II.3. Solution d'orchestration

La plupart des applications construites chez le client se basant sur une architecture en microservices couvrent plusieurs conteneurs, et de même pour les applications monolothiques, ils utilisent parfois des milliers de conteneurs. Lorsque les applications arrivent à cette échelle, nous aurons besoin d'un outil d'orchestration de conteneurs pour que nous puissions les gérer. La gestion à la main n'est pas possible.

Au niveau de cette section, nous allons étudier les solutions d'orchestrations et présenter la solution maintenue dans notre cas et qui sera déployée.

II.3.1. Etude comparative des solutions

Afin de bien choisir notre solution d'orchestration, nous avons commencé par réaliser une étude comparative des solutions existantes. Nous avons comparé les deux solutions les plus utilisées, et les mieux notées qui sont Kubernetes, Docker Swarm.

Le tableau ci-dessous présente l'étude réalisée.

Critères	Kubernetes	Docker Swarm
Définition	Kubernetes automatise le déploiement, la planification de nombreux conteneurs d'application sur des clusters. Il exécute les conteneurs dans des « pods » qu'est l'unité de base et un groupe d'un ou plusieurs conteneurs, Un pod propose un « hôte logique spécifique à l'application ».	Docker Swarm est le système de clustering de Docker, il utilise l'API Docker. L'orchestration d'un cluster avec Docker Swarm nous permet de conserver la compatibilité avec d'autres outils standards.
Évolutivité	Chaque niveau d'application est défini en tant que "pod" et peut être mis à l'échelle lorsqu'il est géré par un déploiement.La mise à l'échelle peut être manuelle ou automatisée.	Les services peuvent être mis à l'échelle à l'aide du Docker Compose. Les services peuvent être globaux ou répliqués.
Mise à l'échelle automatique	La mise à l'échelle automatique à l'aide d'un nombre de pod est définie de manière déclarative au cours de déploiements. Le paramètre CPU utilisé par pod est disponible pour déclencher la mise à l'échelle automatique.	Pas directement disponible. Pour chaque service, nous devons déclarer le nombre de tâches que nous souhaitons exécuter. Lorsque nous augmentons ou réduisons manuellement le gestionnaire Swarm, celui-ci s'adapte automatiquement en ajoutant ou en supprimant des tâches.

Tableau II.8. Kubernetes vs Docker Swarm [8].

Après avoir fait notre étude comparative, nous avons choisi Kubernetes comme solution d'orchestration vu qu'elle admet une grande communauté et qu'elle assure une fléxibilité lors du déploiement des applications.

Dans la section qui suit nous présenterons Kubernetes, ses composants ainsi que l'architecture globale.

II.3.2. Kubernetes

II.3.2.1. Présentation

Kubernetes, également appelé K8s est la solution open source de Google qui assure la gestion de conteneurs pour des environnements de cloud publics, privés et hybrides. Il est conçu pour gérer entièrement le cycle de vie des applications et services conteneurisés en assurant la scalabilité et la haute disponibilité.

En effet il permet d'automatiser le déploiement, la mise à l'échelle, la maintenance, la planification et l'exploitation de plusieurs conteneurs d'applications sur des ensembles de nœuds. Il fournit ainsi un environnement de gestion centré sur le conteneur.

K8s orchestre les infrastructures informatiques, le réseau et le stockage. Cela assure la simplicité de la plate-forme en tant que service (PaaS) et la flexibilité de l'infrastructure en tant que service (IaaS) et permet ainsi la portabilité entre les fournisseurs d'infrastructure [9].

· La figure ci-dessous présente le logo de Kubernetes.

Figure II.8. Logo de Kubernetes

II.3.2.2. Architecture physique

Après avoir présenté l'utilité de Kubernetes, nous allons présenter l'architecture globale d'un cluster Kubernetes, et ceci dans la figure ci-dessous.

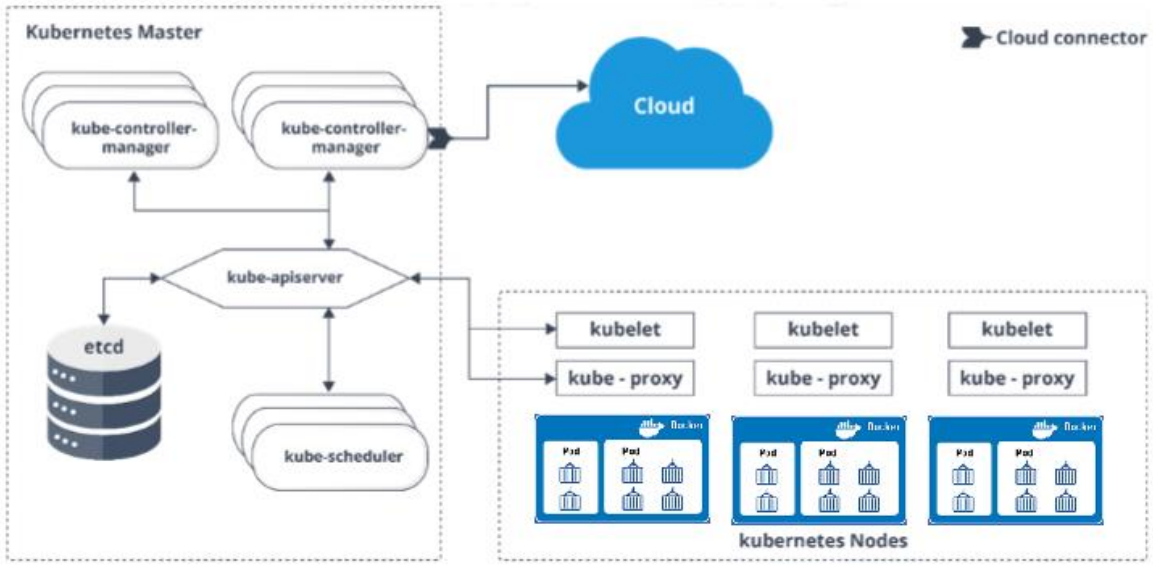

Figure II.9. Architecture physique de Kubernetes

L'architecture de Kubernetes mise en place dans la figure II.9 est composée de :

Nœud Maitre : Le nœud maître est responsable de la gestion du cluster Kubernetes. C'est principalement le point d'entrée pour toutes les tâches administratives

- API Server : C'est est le point d'entrée de toutes les commandes REST utilisées pour contrôler le cluster
- Schedule : Planifie les tâches sur des nœuds esclaves. Il stocke les informations d'utilisation des ressources pour chaque nœud esclave.
- Etcd : Un service de métadonnées.
- Replication controller : Gère la réplication des pods.

Nœuds esclaves : Les nœuds de travail contiennent tous les services nécessaires pour gérer la mise en réseau entre les conteneurs, communiquer avec le nœud maître et affecter des ressources aux conteneurs planifiés.

- Kubelet : Otient la configuration d'un pod du serveur API et s'assure que les conteneurs décrits sont opérationnels.
- Kube-proxy: Agit comme un proxy réseau et un équilibreur de charge pour un service sur un seul noeud de travail.
- Pod : Un pod est un ou plusieurs conteneurs qui s'exécutent logiquement sur des nœuds..

II.4. Solution de déploiement de Kubernetes

Le choix de la solution de déploiement de Kubernetes dépend des ressources physiques et de la flexibilité à assurer. Toutefois, il existe une multitude de solutions, nous pouvons les classer comme suit :

- **Solutions locales :** Il s'agit de l'installation de k8s sur la machine locale.

  C'est ce qui était fait en partie par le client, mais la lenteur de mise en place par rapport au Cloud à empêcher le client de modifier l'architecture de son infrastructure et donc de corriger des erreurs légitimes de conception. Les environnements de développement, de qualification et production ne sont pas étanches par exemple.

- **Solutions Cloud IaaS :** Elles permettent d'offrir une infrastructure dédiée (IaaS). Nous pouvons à cet effet créer et héberger des clusters Kubernetes sur cette infrastructure. Ceci assure plus de liberté et de flexibilité, en contrepartie, il y'a un effort à fournir plus important. C'est la solution où nous voulons avoir plus de contrôle sur les clusters que dans le cas des solutions hébergées.

  C'est ce qui était fait avant en partie par le client, mais ce n'est pas son métier principal, malgré le niveau technique de ses employés certains incidents du réseau d'onduleurs ont causé des pertes des services.

- **Solutions hébergées :** Il s'agit de Kubernetes en tant que service (KaaS). une forme de virtualisation basée sur le conteneur qui fournit l'environnement d'exécution, les outils d'orchestration et toutes les ressources d'infrastructure sous-jacentes via un fournisseur du cloud, nous pouvons citer Amazon Container Service (ACS) ou Google Kubernetes Engine (GKE).

La dernière solution nous intéresse vu que le client n'a pas les moyens pour héberger et superviser les clusters, donc il cherche une solution totalement gérée, de plus elle doit etre automatisée, évolutive et hautement disponible. Il existe plusieurs solutions qui offrent ce service, nous allons donc réaliser une étude comparative des solutions on premise.

II.4.1. Conteneurs en tant que service 'CaaS'

Pour suivre le modèle classique du cloud computing, CaaS pourrait être placé entre IaaS et PaaS. Cependant, parmi ces modèles de service, Container-as-a-Service se distingue par une approche fondamentalement différente de la virtualisation: utiliser la technologie de conteneur.

II.4.2. Choix de la solution de déploiement 'CaaS'

Nous avons mis la choix de Google comme fournisseur de notre solution pour plusieurs raisons

Le tableau ci-dessous décrit comment google assure la pérennité de notre solution

	Avantages	Disavantages
Google Kubernetes Engine (GKE)	-Intégration complète avec d'autres produits Google -Déploiement de conteneurs sur hybride et multi-nuages possible	En ce qui concerne les clusters de 6 nœuds ou plus, Google facture des frais d'utilisation pour le moteur de conteneur en plus des coûts des opérations arithmétiques.
Amazon Container Service (ACS)	- Intégration complète avec d'autres produits AWS	- Le déploiement du conteneur est limité aux instances Amazon EC2. - Les outils Open Source communs tels que Kubernetes ne sont pas pris en charge.

Tableau II.5. GKE vs ACS

En ce qui conserne le coût, Google et Amazon ont des projets différents en termes de tarification de leurs services CaaS. Chez Google les utilisateurs peuvent accéder gratuitement à la gestion des conteneurs pour les clusters comportant jusqu'à cinq instances de moteur informatique (nœuds). La fourniture de services en nuage (CPU, mémoire, etc.) ne représente que des coûts. Si les utilisateurs souhaitent exécuter des conteneurs sur des grappes plus

grandes (six instances ou plus), Google facture également des frais d'utilisation pour le moteur de conteneur: le tarif est indiqué à l'heure pour une grappe [14].

II.4.3. Google Kubernetes Engine 'GKE'

Google a intégré un service de conteneur hébergé avec le moteur de conteneur Google (GKE) dans le cloud. Le composant principal du service CaaS est constitué des outils d'orchestration Kubernetes .

L'intégration de Kubernetes dans GKE offre aux utilisateurs les fonctions suivantes pour orchestrer des applications de conteneur [10]:

- **Localistation automatique:** Kubernetes localise automatiquement les conteneurs en fonction des besoins en ressources et des contraintes afin que le cluster soit toujours chargé de manière optimale. Cela évite que la disponibilité des applications de conteneur ne soit compromise.
- **Equilibrage de charge:** Kubernetes propose deux modes de découverte de service: les services peuvent être détectés à l'aide de variables d'environnement et d' enregistrements DNS . L'équilibrage de charge entre différents conteneurs peut être effectué à l'aide de l'adresse IP et des noms DNS.
- **Mise à l'échelle horizontale:** avec Kubernetes, les applications peuvent être agrandies ou réduites à la demande, soit manuellement, soit à l'aide de la ligne de commande de l'interface utilisateur graphique, ou automatiquement, en fonction de l'utilisation de la CPU.
- **Gestion des identités et accès:** GKE de IAM est mis en œuvre à l' aide des comptes Google et soutient d' autres autorités basées sur les rôles.
- **Orchestration du stockage:** Kubernetes autorise le stockage automatique sur divers systèmes de stockage, qu'il s'agisse de stockage local, de stockage dans le cloud public (via GCP ou AWS) ou de systèmes de stockage réseau tels que NFS, iSCSI, Gluster, Ceph ou Flocker.

II.5. Solution de livraison continue

Kubernetes est compatible avec la majorité des outils CI / CD, ce qui permet aux développeurs de tester, de déployer des versions dans Kubernetes et de mettre à jour des

applications sans interruption. Jenkins est l'un des outils de CI / CD les plus populaires à l'heure actuelle

Nous avons été amenés à travailler avec Jenkins pour plusieurs raisons :

- Le client a déjà énormément de Jenkinsfile qui seraient longs à migrer.
- Tous les modules existent dessus ce qui assure la pérennité de la solution
- Jenkins est également assez facile à configurer, à modifier et à étendre. Il déploie le code instantanément, génère des rapports de test. Jenkins peut être configuré en fonction des exigences d'intégration continue et de livraison continue.
- **Evolutive :** L'un des aspects les plus forts de Jenkins est qu'il dispose d'une fonction de mise à l'échelle presque prête à l'emploi. La mise à l'échelle Jenkins est basée sur le modèle maître / esclaves, dans lequel nous avons plusieurs instances d'agent (appelées esclaves) et une instance principale Jenkins (appelée maître), qui est principalement responsable de la répartition des tâches entre les esclaves.

Il y a beaucoup d'options disponibles pour mettre en œuvre la mise à l'échelle Jenkins. Kubernetes est l'un des meilleurs outils de gestion de solutions évolutives grace à Helm gestionnaire de packages de Kubernetes, pratiquement tout peut être placé à l'intérieur des conteneurs. Jenkins peut l'être aussi, et c'est pourquoi Kubernetes est une très bonne option à utiliser pour nos besoins.

II.5.1. Helm

C'est le gestionnaire de packages pour Kubernets. Il assure le déploiement des applications complètes sur Kubernetes en utilisant un système de templating et de dépendances. Son but est d'éviter la duplication et avoir une arborescence cohérente des fichiers de configurations. Il permet ainsi de définir, d'installer et de mettre à niveau même les applications les plus complexes sur Kubernetes.

Figure II.10. Logo de Helm

Dans notre cas, nous avons utilisé Helm pour déployer Jenkins à partir du dépôt Charts.

Un chart est une collection de fichiers de configuration de déploiement.

Jenkins Helm chart nous offre :

- Possibilté pré-configurer l'installation de Chart (Installation des plugins, monter les sockets Docker, attacher un volume persistent..)
- Mise en place de pod de déploiement Jenkins Maitre
- Exposition de service Jenkins à l'extérieur à travers une adresse IP de LoadBalancer
- Fourniture à la demande d'agent Jenkins (JNPL)

Pour faciliter la coordination avec le cluster Kubernetes, le gestionnaire de packages Helm comprend deux composants principaux: le client Helm qui gère les charts de déploiement et communique avec l'application serveur Tiller qui est installé au niveau du cluster Kubernetes.Tiller recoit la demande de déploiement de Helm via gRCP puis il transmet la configuration à l'API Kubernetes.

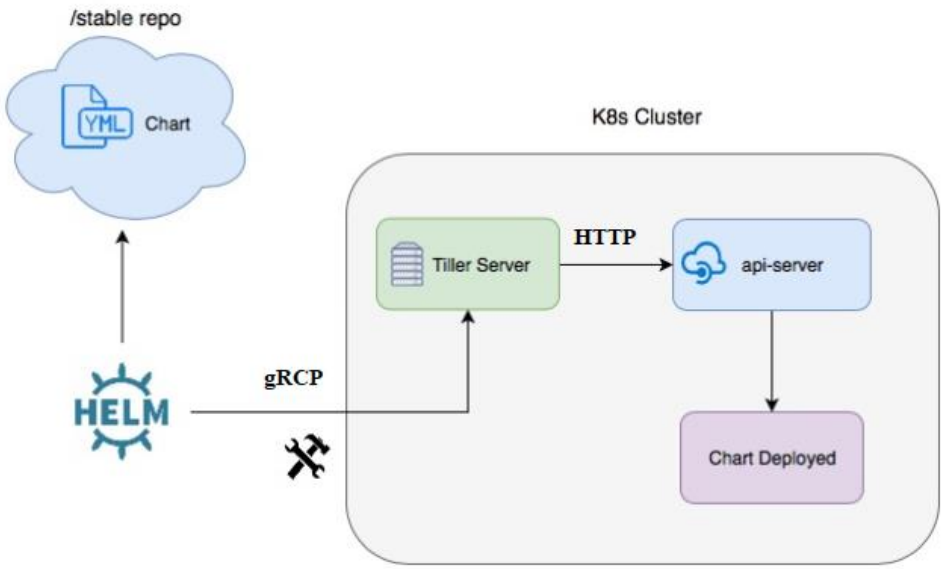

Figure II.11. Architecture de Helm

II.5.2. Flux de travail

La figure ci-dessous représente l'interaction entre Jenkins et Kubernetes

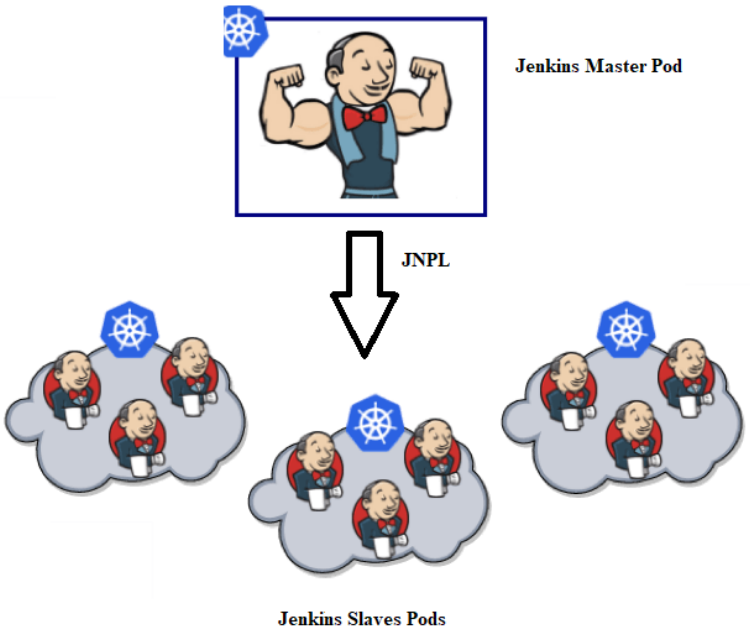

Figure IV. 12. Flux de travail Jenkins [11]

- Lorsque le travail est déclenché à partir du maître, il recherche les détails de la configuration de l'esclave.
- En fonction de la configuration donnée en maître à l'aide de l'image esclave JNLP Jenkins, un pod esclave est créé à l'aide du protocole JNLP (Java Network Launch Protocol), celui qui va établir la connexion avec les binaires de Docker et affecté à la construction générée à partir du maître jusqu'à ce que la construction soit terminée.
- Si la construction est réussie, l'esclave recherchera d'autres planificateurs de construction pendant un certain temps. S'il n'y a pas de réponse du maître, l'esclave publiera les résultats de construction existants sur le maître et se terminera.

II.6. Solution d'emplacement des images Docker

Étant donné que nous allons utiliser Google Kubernetes Engine, le registre de conteneurs Google GCR, constitue une solution naturelle pour notre registre privé afin de stocker en ligne les images Docker.

- ✓ Elle est déjà prête et ne demande pas d'administration
- ✓ Elle est privée, performante et bénéficie de contrôles d'accès.

En gros, c'est surtout dans l'autre sens qu'il faut le voir, installer soi-même un registre pose des problèmes de maintenabilité, utiliser un service de stockage en ligne coûte de l'argent et pose des risques de sécurité additionnels.

II.7. Solution d'automatisation Terraform

Notre solution consiste à livrer automatiquement la plateforme des conteneurs, dans ce cas-là nous avons besoins d'un outil d'automatisation afin de définir toute l'infrastructure dédiée.

Terraform est un outil permettant la création des infrastructures IT sur le Cloud fonctionnant en mode infrastructure en tant que code au lieu de procéder à un traitement manuel répétitif et pénible.

Nous avons choisi Terraform parce que :

- On peut avoir des ressources multi-cloud tous ensemble
- Facile à intégrer les supports Kubernetes et Helm
- On a la main de gérer un nombre important d'infrastructures et de composants GCP
- Maintenue par une grande communauté ainsi que le fournisseur GCP pour assurer sa pérennité

En effet, c'est un outil qui lit une description de ressources Cloud (serveurs, sous-réseaux, API, user..) par le biais des scripts écrits en HCL, compare la description a l'existant, et s'il voit qu'il y a des changements à faire, il appelle les API Cloud et fait les changements nécessaires.

II.8. Architecture globale

Suite à l'étude faite de déférents composants qui existent sur le marché, nous présentons dans la figure ci-dessous l'architecture logicielle globale de la solution proposée

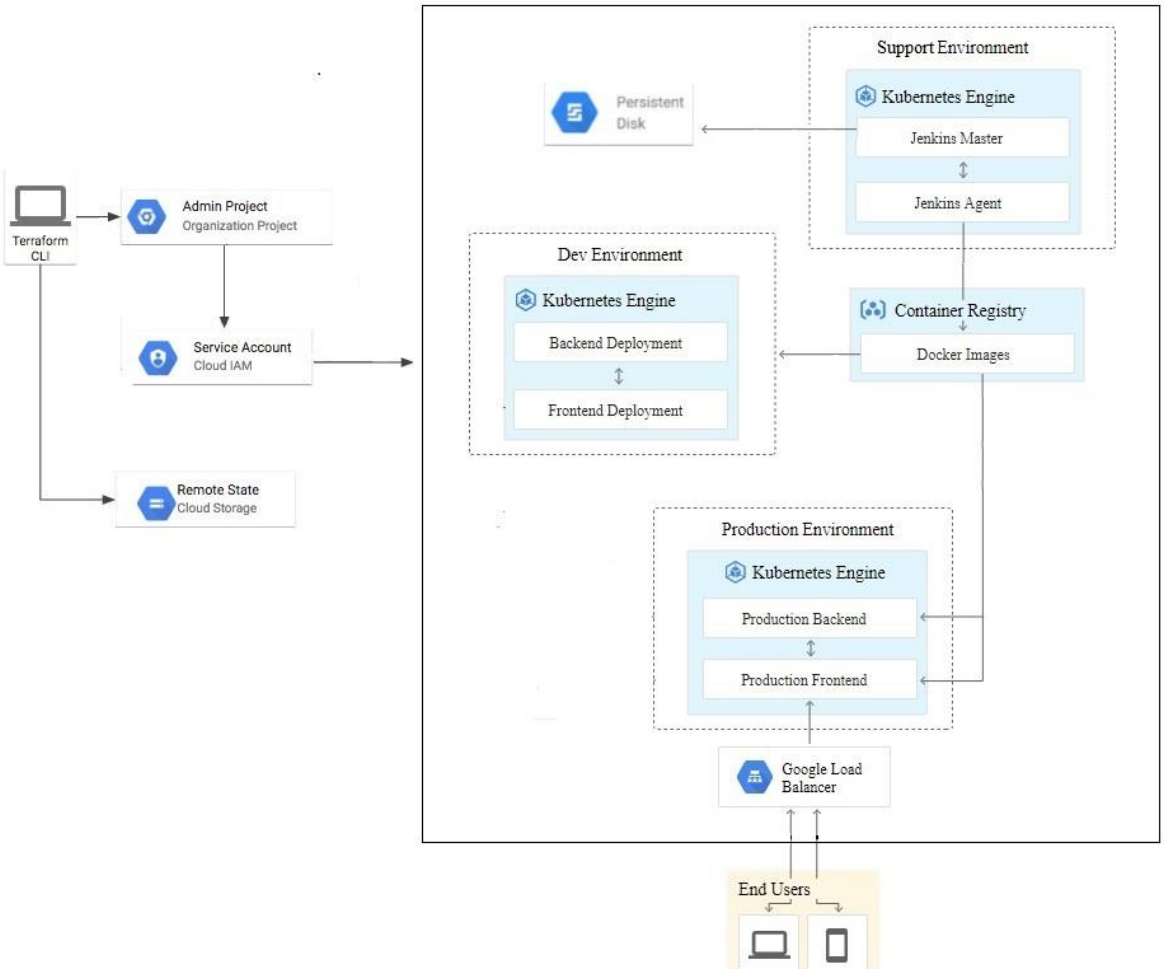

Figure II. 13. Architecture globale de projet

Conclusion

Ce chapitre a présenté une étude conceptuelle de notre solution. Il nous a permis de capturer les besoins techniques, choisir les technologies à implémenter et réaliser la conception générique de la solution. Dans le chapitre suivant, nous entamerons la phase cocuptuelle de nore solution.

Chapitre III. Conception

Plan

Introduction

Afin d'identifier les fonctionnalités que notre environnement doit satisfaire, nous extrayons les spécifications fonctionnelles et non fonctionnelles et nous présentons les exigences que nous visons à offrir via un modèle de cas d'utilisation et de séquence.

III.1. Identification des acteurs

Un acteur est une entité externe qui interagit avec le système. En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin [12].

Dans notre système, nous avons toris acteurs qui sont :

- **Administrateur thenique:** Chargé de créer les projets sur GCP, gérer les utilisateurs et créer les comptes des services.
- **Administrateur Cloud :** - Automatiser la livraison d'une plateform de contenneurs GKE
  - Automatiser le déploiement d'un serveur d'intégration
  - Automatiser le déploiement des applications.
- **Integrateur :** Gérer la configuration des builds d'intégration.

III.2. Besoins fonctionnels

Dans la phase d'identification des besoins fonctionnels du projet, nous nous sommes basés sur les informations détenues par l'entreprise sur les problèmes rencontrés dans la solution existante.

Après l'analyse de ces problèmes, nous avons pu déduire les fonctionnalités pouvant répondre aux besoins de l'entreprise. Les besoins fonctionnels auxquels doit répondre notre solution sont les suivants :

- Développer l'infrastructure en tant que code afin d'automatiser la mise en place de 3 clusters de conteneurs GKE (prod, pre-prod et support) sur GCP.
- Configurer les zones DNS et les plages VPC pour les 3 clusters Kubernetes.
- Mise en place d'un serveur de déploiement Tiller pour faire tourner les chartes Helm.

- Déploiement du serveur d'intégration Jenkins en utilisant Helm au niveau du cluster support.
- Spécifier la configuration de l'installation de serveur Jenkins
- Monter un volume persistant au pod de déploiement Jenkins.
- Lancer le build de l'un des applications de notre client
- Intégrer Google Registry au niveau de serveur Jenkins pour stocker les conteneurs de build
- Automatiser le déploiement de l'application conteneurisé

Les fonctionnalités globales sont illustrées dans le diagramme de cas d'utilisation qui est présenté dans la figure ci-dessous.

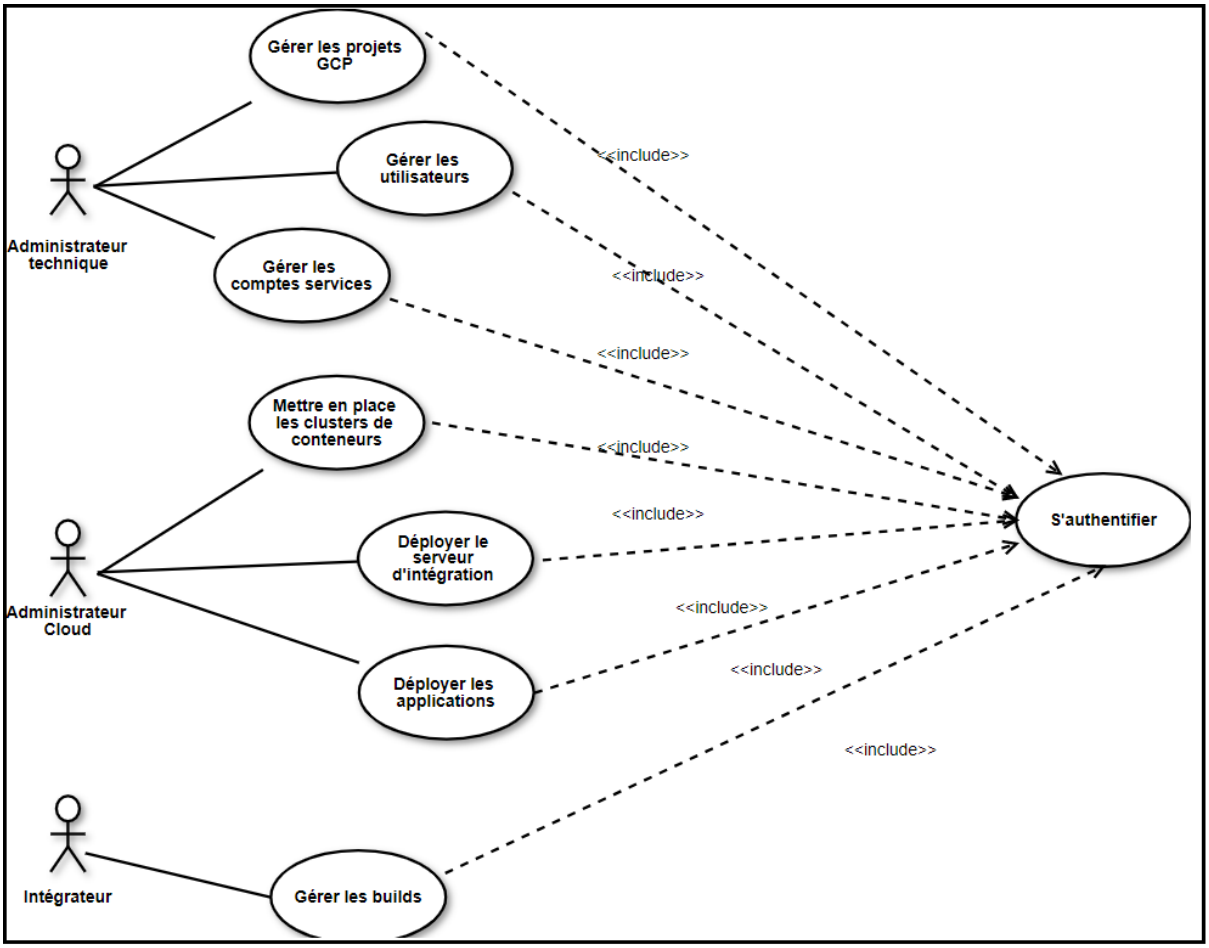

Figure III.14. Diagramme de cas d'utilisation globale

III.3. Besoins non fonctionnels

Les besoins non fonctionnels auxquels doit répondre notre solution sont :

- **Sécurité :** Il faut sécuriser l'accès à l'infrastructure et aux conteneurs.
- **Adaptabilité :** les solutions doivent pouvoir être étendues, complétées ou développées facilement à mesure que les besoins de l'entreprise évoluent.
- **Facilité de maintenance :** la maintenance du cluster et la mise à jour des composants doivent être de complexité réduite.
- **Élasticité :** possibilité de mettre en ligne ou de retirer des ressources de calcul (CPU, mémoire, réseau) à mesure que la charge de travail augmente ou diminue.
- **Interopérabilité :** si nous voulons utiliser les services de plusieurs fournisseurs de cloud, nous devrions avoir la possibilité de déplacer des charges de travail entre fournisseurs de cloud.

III.4. Scénarios des cas d'utilisation

Après avoir cité les différentes fonctionnalités offertes par la solution, nous allons, dans cette section, les modéliser à travers des diagrammes de cas d'utilisation d'UML qui est un langage de modélisation qui convient de manière adéquate aux approches objets, en illustrant les interactions entre les acteurs et le système par des diagrammes de séquences système.

Un cas d'utilisation est un ensemble de séquence d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur particulier [12].

III.4.1. Raffinement du cas d'utilisation " Mettre en place les clusters"

La figure III.13, représente le diagramme de cas d'utilisation « Mettre en place les clusters».

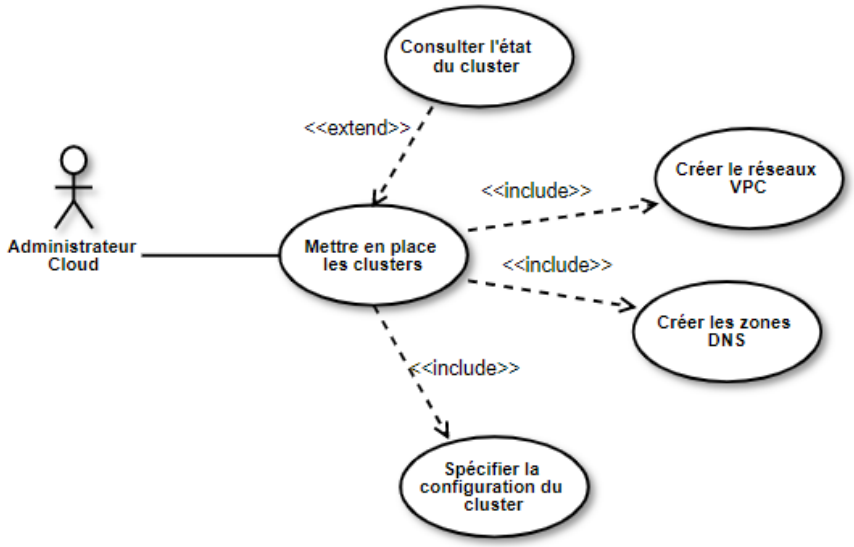

Figure III.13. Cas d'utilisation « Mettre en place les clusters »

L'administrateur de nuage automatise la mise en place des clusters Kubernetes, specifie ses configurations et la gestion de ses ressources à travers GKE API. Ainsi, il a le droit de consulter et de gérer la totalité du nuage via la console Google Cloud.

Le tableau suivant représente l'acteur et la description de scénario du cas d'utilisation « Mettre en place les clusters».

Cas d'utilisation	Mettre en place les clusters
Acteur	Administrateur Cloud
Pré condition	Acteur authentifié et autorisé
Post condition	Environnement de conteneurisation crée et configuré
Scénario principale	• Ecrire la configuration dédiée à la mise en place de 3 clusters Kubernetes (prod, preprod et support) • Ecrire la configuration de topologie résaux VPC • Ecrire la configuration des zones DNS • Lancer les scripts d'automatisation
Extension	Consulter l'état de l'infrastructure

Tableau III.6. Description de cas d'utilisation « Mettre en place les clusters »

Afin de mieux interpréter le processus décrit dans le tableau ci-dessus, nous présentons dans ce qui suit le diagramme de séquences système qui décrit le scénario nominal du cas d'utilisation «Mettre en place les clusters», ce diagramme présente les différents messages échangés entre les différents services de Google Cloud afin d'automatiser la configuration et la mise en place des grappes Kubernetes. En d'autres termes, il installe un environnement adéquat pour assurer le bon fonctionnement des conteneurs des applications.

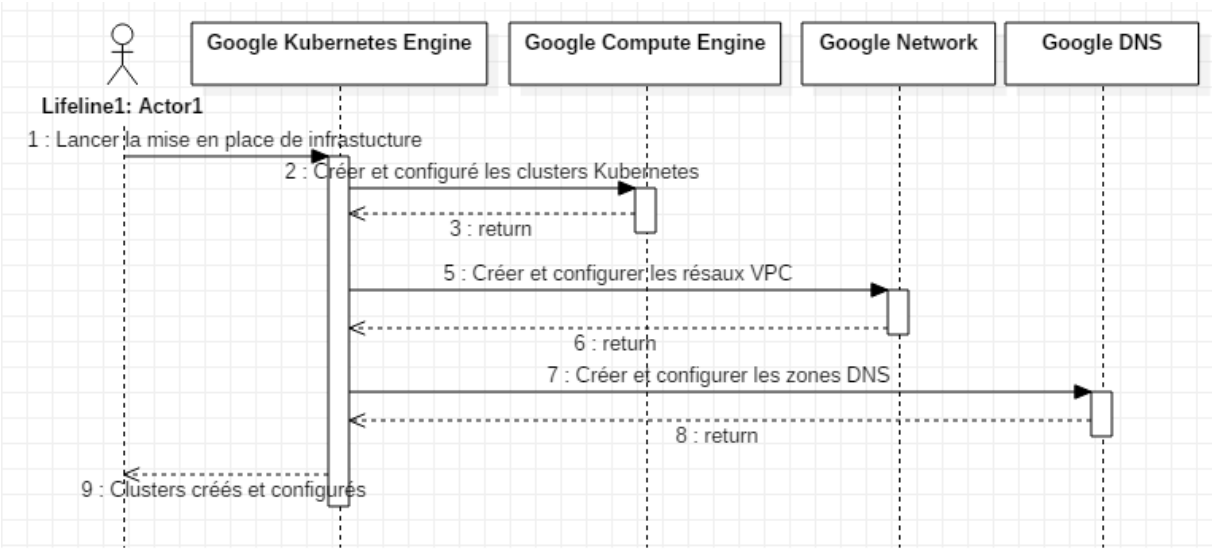

Figure III.14. Diagramme de séquence « Mettre en place les clusters »

III.4.2. Raffinement du cas d'utilisation " Déployer une application "

La figure III.15, représente le diagramme de cas d'utilisation « Déployer une application ».

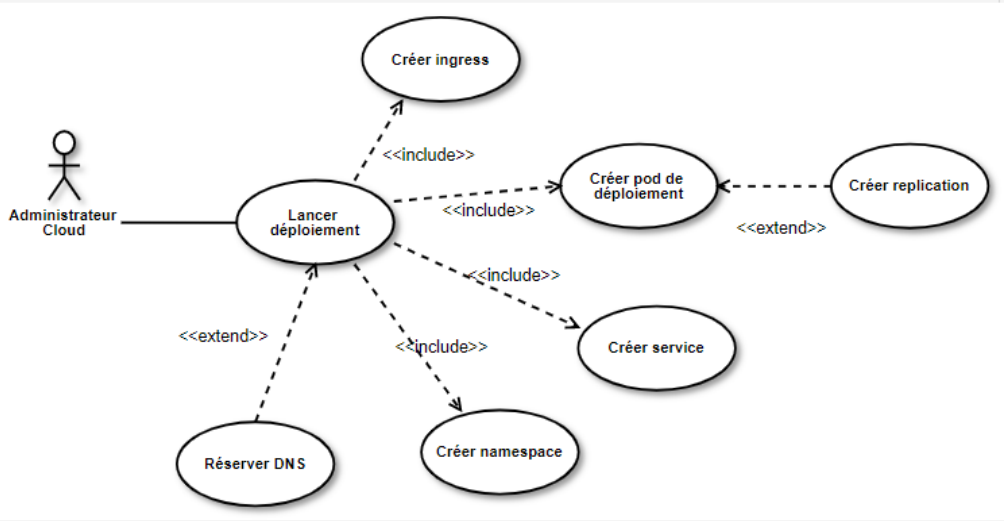

Figure III.15. Diagramme de cas d'utilisation « Déployer une application »

Cas d'utilisation	Déployer une application
Acteur	Administrateur Cloud
Pré condition	Acteur authentifié et autorisé
Post condition	Application déployé
Scénario principale	• Ecrire la configuration dédiée à la mise en place automatique de déploiement d'une application conteneurisé • Lancer les scripts d'automatisation
Extension	• Répliquer le déploiement • Réserver un DNS

Tableau III.7. Description de cas d'utilisation « Déployer une application »

Le déploiement d'une application conteneurisé requiert plusieurs étapes. L'administrateur doit commencer par créer un namespace pour le projet, puis, il doit créer le pod de déploiement de conteneur ainsi que ses réplications, ensuite, il doit créer un service pour le pod deployé. Enfin, il doit créer un ingress de la conteneur déployée afin d'éxposer le service dédié à l'exterieur à travers une adresse IP gérer par l'équilibreur de charge Kubernetes, il peut aussi résérver un DNS à cette adresse. Ces étapes sont présentées sur le diagramme de séquence ci-desoous qui porte plus de détails sur les interactions entre les différents composants de Kubernetes cluster.

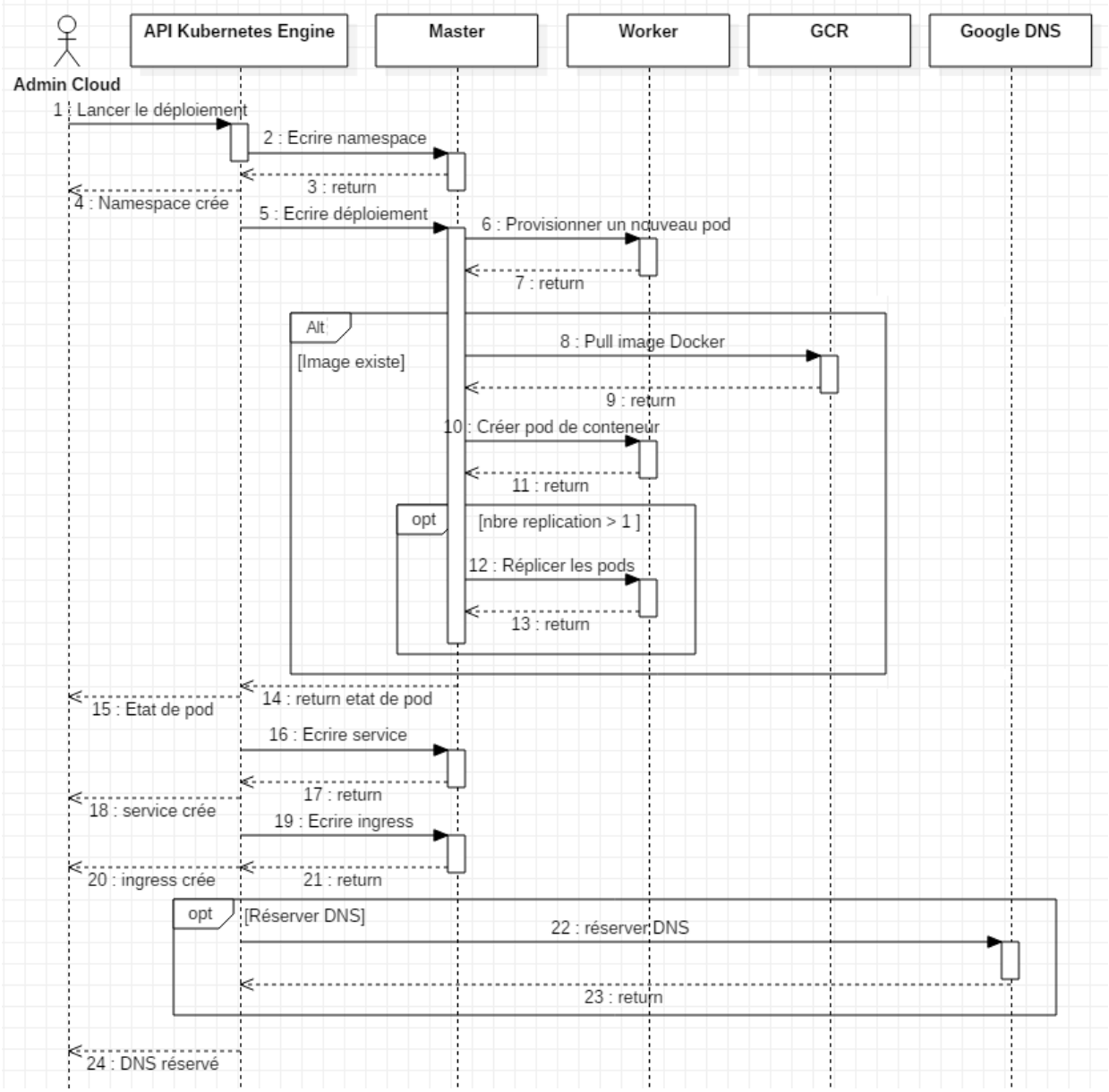

Figure III.16. Diagramme de séquence « Déployer une application »

III.4.3. Raffinement du cas d'utilisation " Déployer serveur d'intégration"

Une fois notre infrastructure est mis en place, l'administrateur automatise le déploiement de serveur d'intégration Jenkins sur la grappe support.Ainsi, il a le droit d'ajouter des spécifications pour le Chart Jenkins afin d'assurer que l'intégrateur aura la main sur les différentes procédures possibles en relation avec les conteneurs. De plus il faut créer un volume persistant et lui monter sur le pod de déploiement afin d'éviter le perd des données.

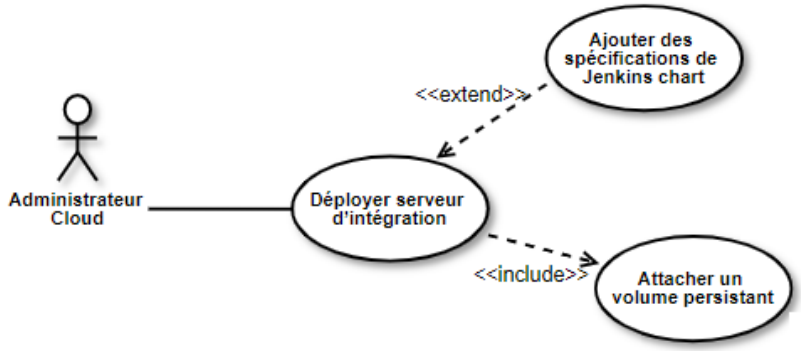

Figure III.172. Diagramme de cas d'utilisation « Déployer le serveur d'intégration »

Le tableau suivant représente l'acteur et la description de scénario du cas d'utilisation « Déployer le serveur d'intégration ».

Cas d'utilisation	Déployer le serveur d'intégration
Acteur	Administrateur Cloud
Pré condition	Acteur authentifié et autorisé
Post condition	Jenkins configuré et déployé
Scénario principale	• Ecrire la configuration dédiée à la mise en place automatique de Jenkins • Claim un volume persistant au pod de déploiement de Jenkins • Lancer les scripts d'automatisation
Extension	Ajouter des spécifications à la charte Jenkins

Tableau III.8. Description de cas d'utilisation « Déployer le serveur d'intégration »

Helm utilise le serveur Tiller qui doit etre déployer au niveau de cluster support, celui qui va s'intéragir avec les API de cluster Kubernetes pour éxecuter le déploiement de charte Helm.

La figure III.18 décrit les différents messages échangés entre les différents composants de cluster Kubernetes

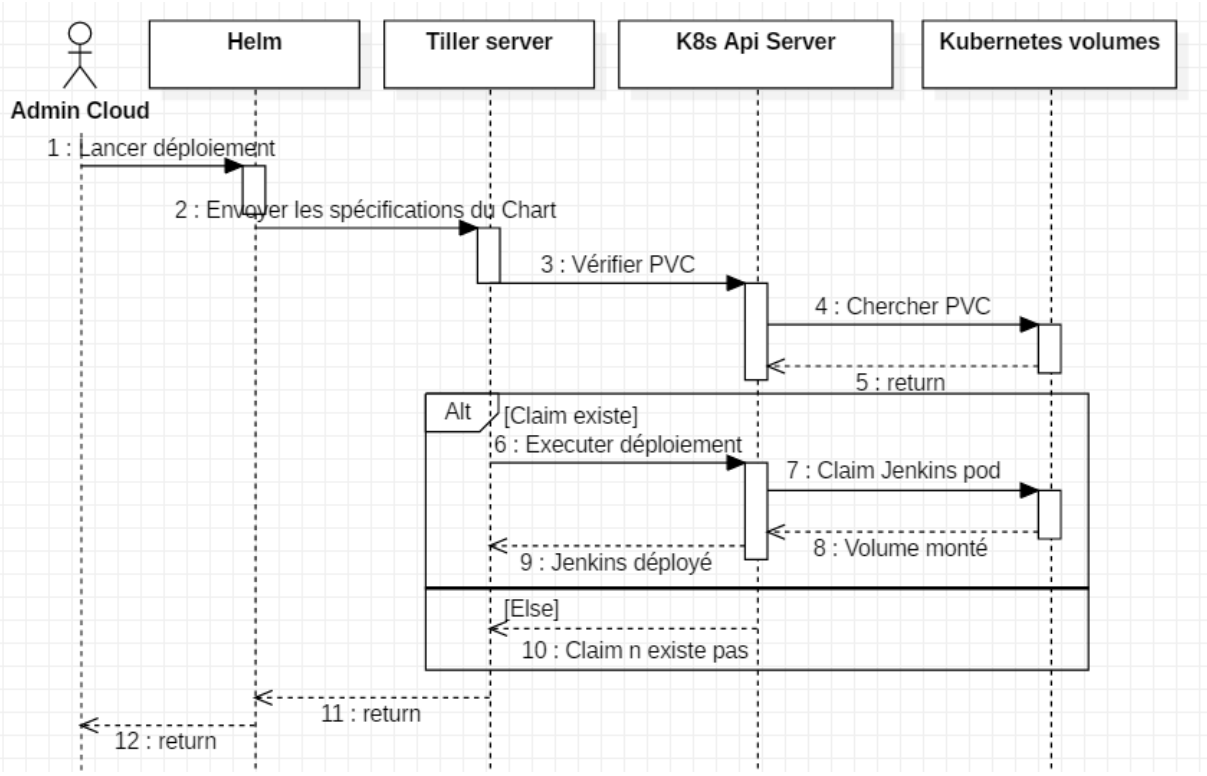

Figure III.18. Diagramme de séquence « Déployer le serveur d'intégration ».

III.4.4. Raffinement du cas d'utilisation "Lancer le build"

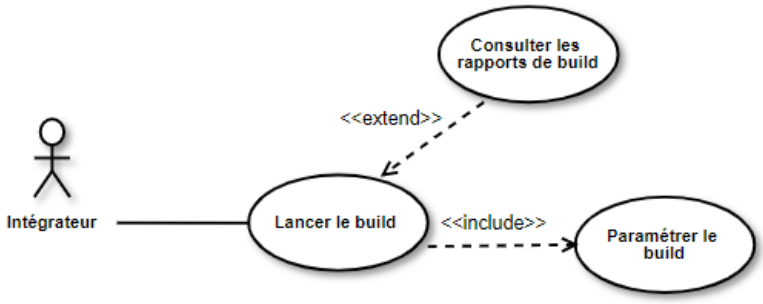

Figure III.19. Diagramme de cas d'utilisation « Lancer le build »

Le tableau suivant représente l'acteur et la description de scénario du cas d'utilisation

Le tableau suivant représente l'acteur et la description de scénario du cas d'utilisation

« Lancer le build ».

Cas d'utilisation	Lancer le build de conteneur
Acteur	Administrateur Cloud
Pré condition	Acteur authentifié et autorisé
Post condition	Succès de build
Scénario principale	• Connecter à l'interface Jenkins • Paramétrer le build • Intégrer le code source de l'application • Intégrer le registre privé GCR • Lancer le build
Extension	Consulter les rapports de build

Tableau III.9. Description de cas d'utilisation « Lancer le build »

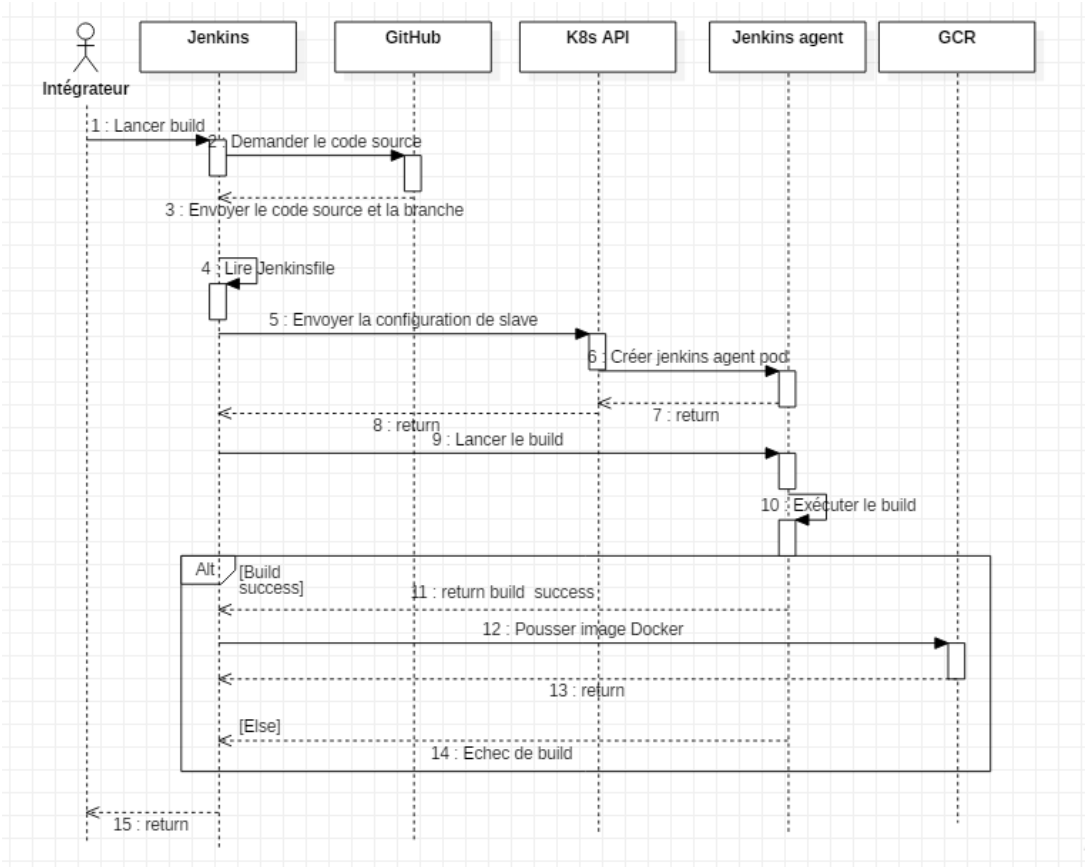

Figure III.20. Diagramme de séquence « Lancer le build »

Une fois le pipeline de build est lancé, Jenkins va d'abord chercher le code source du projet de la part de serveur GitHub, Ensuite il envoie les configurations nécessaire de lancement de l'agent à travers API server de cluster Kubernetes, Après l'agent exécute le build de l'application et faire la publication des résultats de construction existants sur le maître et se terminera.

III.5. Etude conceptuelle

Dans cette section nous présentons les processus métiers utilisée pour mapper le workflow de notre projet afin de montrer les changements positifs sur le modèle de travail actuel.

III.5.1. Présentation du workflow

Le workflow est une technologie utilisée pour assister, automatiser et surveiller le traitement de groupes de tâches complets ou partiels. Un workflow est la série des activités nécessaires à la réalisation d'une tâche.

Pour clarifier les choses, la solution que nous souhaitons mettre en œuvre repose essentiellement sur les concepts de workflow : Nous fournissons un environnement qui automatise le processus de livraison continue via une plate-forme flexible et cohérente, pour faciliter la surveillance des tâches en cours et des activités. Le tableau 11 illustre les principales fonctionnalités fournies via notre processus de flux de travail lié à notre solution cible

Livraison continue	Fourniture d'une plateforme de livraison continue
Orchestration	La solution vise à optimiser et normaliser les processus

Tableau III.10. Caractéristiques principales du processus de workflow du projet

III.5.2. Analyse du système

Pour visualiser plus clairement les concepts mentionnés dans ce tableau, nous nous référerons à un diagramme qui révèle en détail les processus et les activités de base de ce workflow.

III.5.2.1. Workflow livraison continue

Le diagramme d'activité illustré dans la Figure 21, explique le fonctionnement et l'interaction entre les différents composants du pipeline intégration continue. Une fois qu'un nouveau Job est lancé, GitHub va déclencher notre pipeline, la première étape c'est de créer un exécuteur Jenkins afin de créer, ensuite tester et pousser les images de l'application sur le registre privé du projet.

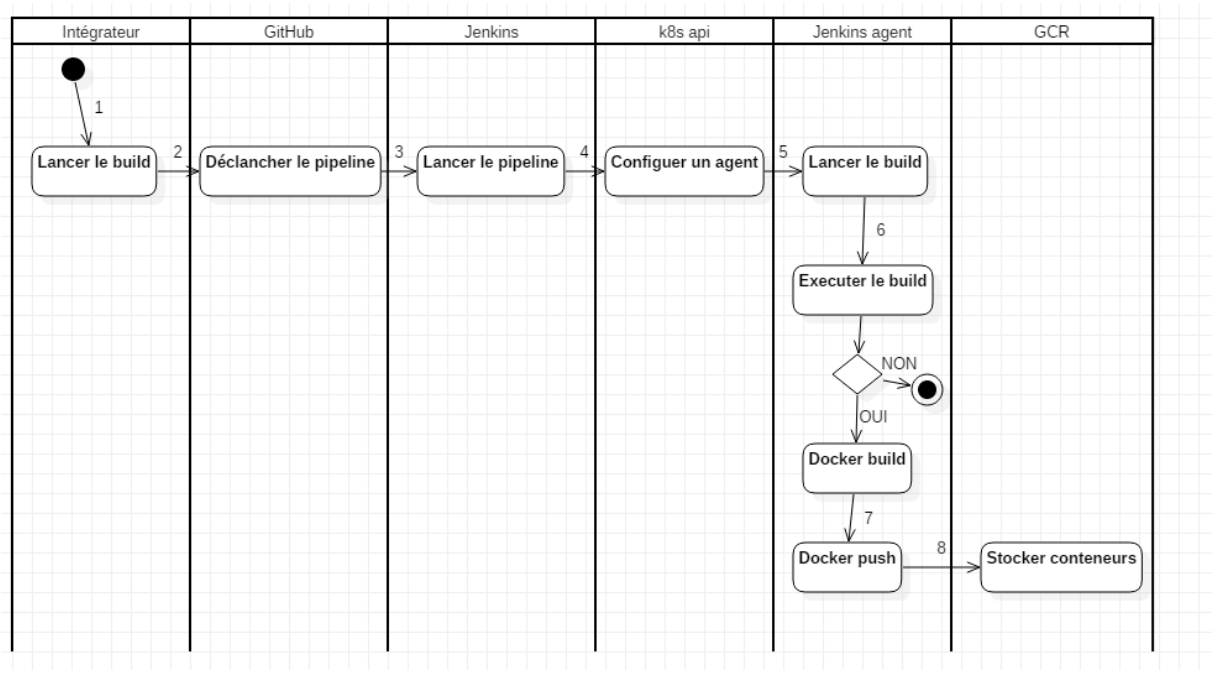

Figure III.21. Diagramme d'activité « Livraison continue »

III.5.2.1. Workflow orchestration

Si les images sont poussées avec succès, la procédure du déploiement est la suivante : le nœud master Kubernetes choisit le nœud slave au sein duquel le Pod sera créé. Ce choix, se base sur les ressources nécessaires pour la création du Pod et les ressources disponibles dans les nœuds du cluster. Une fois le nœud choisi, une adresse IP interne est alors assignée au Pod, cette adresse sera reconnue uniquement à travers le cluster Kubernetes. Ensuite, Kubernetes instancie l'image docker qu'il a préalablement récupéré du registre docker à la suite du build réussi, ainsi un conteneur est créé. Kubernetes expose le service du conteneur et lui crée une route afin qu'il soit accessible pour les utilisateurs.

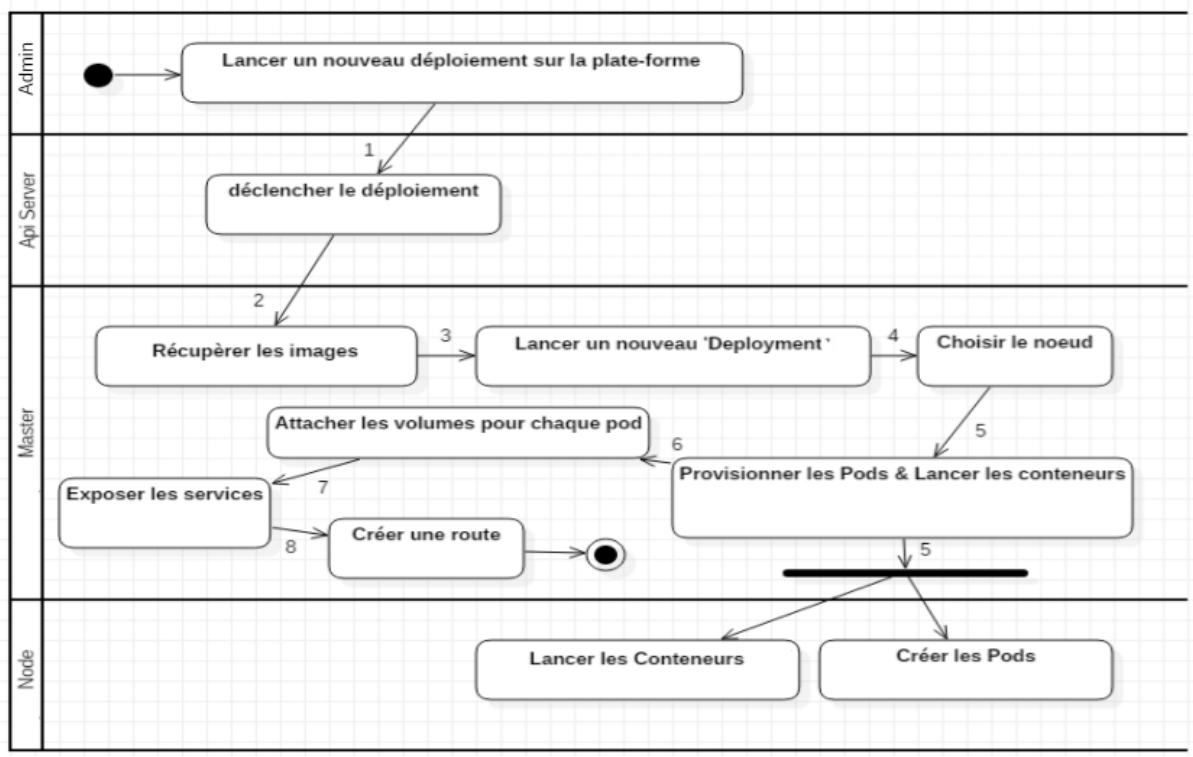

Figure III.22. Diagramme d'activité « Orchestration »

Conclusion

Les cas d'utilisations et les diagrammes de séquence présents ci-dessus couvre seulement l'aspect statique de notre solution, or nous devons automatiser tous le processus, en d'autre termes éliminer les configurations manuelles et les tâches répétitives. Mais avant d'entamer notre modèle proposé de notre solution, nous Dans le dernier chapitre nous allons décrire en détail la phase d'implémentation de chaque solution.

Chapitre IV. Réalisation

Plan

Introduction

La satisfaction des besoins et des exigences requises se voit dans la réalisation du projet. C'est dans ce chapitre que nous exposons le fruit de notre étude après le traitement de la partie conceptuelle du projet. Nous allons commencer en premier lieu par la présentation de l'environnement chez le fournisseur Google. Par la suite, Nous décrivons les différentes étapes de notre projet avec quelques interfaces qui concordent avec les fonctionnalités attendues.

IV.1. Préparation de l'environnement

IV.1.1. Préparation des projets

Google Cloud organise toutes les ressources dans des dossiers et des projets. Cela se transforme en une belle hiérarchie comme celle-ci:

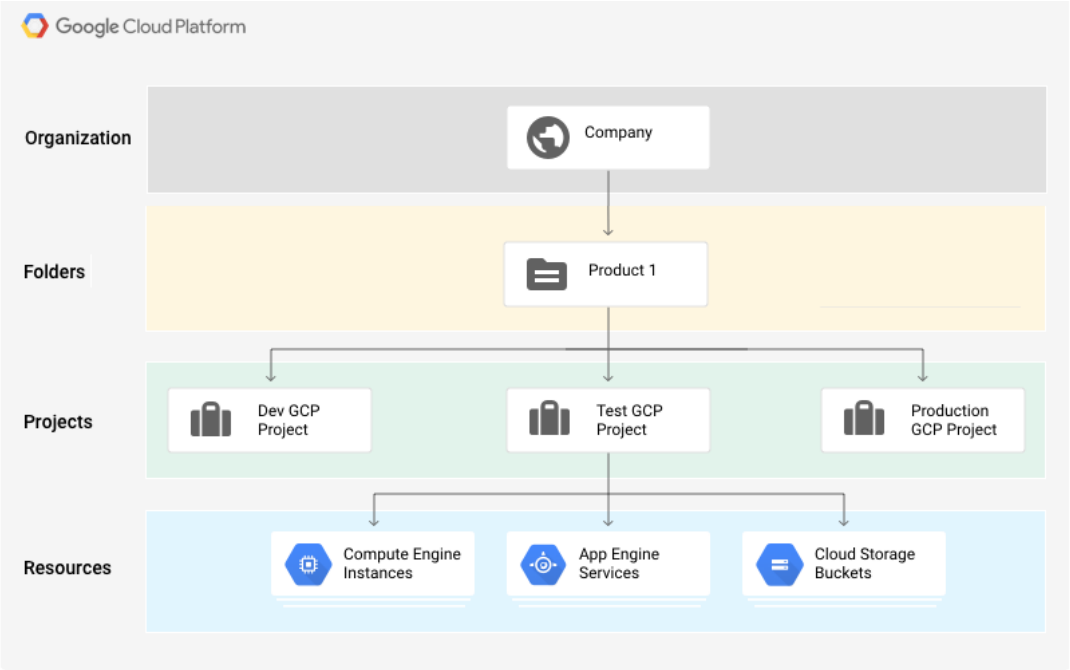

Figure IV.22. Hiérarchie GCP

Dans le cas de notre solution nous avons besoins de créer 3 projets.

- Production operations : Environnement de support
- Production applications : Environnement de production
- No production applications : Environnement de pre-prod

La raison en est que Terraform doit travailler dans le contexte d'un projet, Terraform ne peut pas nous aider à configurer cela. Nous devrons donc effectuer cette partie manuellement à l'aide de la console Google Cloud.

IV.1.2. Mise en relation avec GCP

Terraform interagit avec Google Cloud par le biais de son API. Et pour des raisons de sécurité, il doit d'abord être activé, dans ce cas-la, il faut activer l'API GKE et créer un compte de service pour Terraform et lui attribuer deux roles :

- **Kubernetes Engine Admin :** afin que Terraform avoir l'autorité de créer les ressources dédiée de l'infrastructure sur Google Cloud.
- **Storage Admin :** utilisé par Terraform lui-même, en tant que backend pour garder l'état de l'infrastructure dans un fichier .tfstate

Par la suite, il faut télécharger le fichier JSON qui contient les détails de compte de service ainsi qu'un clé privé pour établir la connexion avec GCP et avoir l'autoriter d'utiliser ses services.

IV.1.3. Initialiser le projet

Terraform init : Cette commande effectue différentes étapes d'initialisation afin de préparer ou mettre à jour le répertoire de travail à utiliser, celon les modifications apportées à la configuration. Cela lira le code source, trouvera les plugins nécessaires, les téléchargera et les installera.

```
souhail@souhail:~/devops-infra-base/terraform$ terraform init

Initializing the backend...

Successfully configured the backend "gcs"! Terraform will automatically
use this backend unless the backend configuration changes.

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "helm" (terraform-providers/helm) 0.10.2...
- Downloading plugin for provider "google-beta" (terraform-providers/google-beta
) 2.14.0...
- Downloading plugin for provider "google" (terraform-providers/google) 2.14.0..
.
- Downloading plugin for provider "kubernetes" (terraform-providers/kubernetes)
1.9.0...

* provider.helm: version = "~> 0.10"
* provider.kubernetes: version = "~> 1.9"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

Figure IV.23. Inisialiser le projet

Dans la figure ci-dessus, on peut visualiser la préparation de différents provider de Terraform qui sont nécessaire de mettre notre solution en place, ainsi que l'inisialisation du backend qui consiste à sauvgarder l'état de l'infrastructure

Ensuite, nous utilisons une combinaison de terraform validate, terraform plan et terraform apply pour implémenter notre solution.

IV.2. Mise en place de l'infrastructure

Dans la section qui suit, nous allons présenter les résultats d'implémentation de chaque solution au niveau de notre infrastructure. nous avons déjà choisi Google Kubernetes Engine (GKE) comme solution de déploiement de Kubernetes.

A cet effet, nous avons commencé par la mise en place des grappes Kubernetes.Puis, nous avons mis la configuration nécessaire des nœuds pour chaque cluster, ainsi que la configuration du réseaux VPC, les régles des par-feux et les zones DNS.

IV.2.1. Mise en place des clusters Kubernetes avec GKE

Notre solution consiste à la mise en place 3 grappes de conteneurs Kubernetes :

- **be-production-operations :** Services pour l'exécution des opérations. Il s'agit d'un projet de production, mais les services sont également utilisés pour gérer des environnements non productifs.
- **be-production-applications :** Les applications elles-mêmes, par exemple sites Web, bases de données, etc.
- **be-no-production-applications :** Toujours des applications mais dans un environnement non productif.

La figure ci-dessous présente l'ensemble des nœuds déployés au niveau du cluster de support.

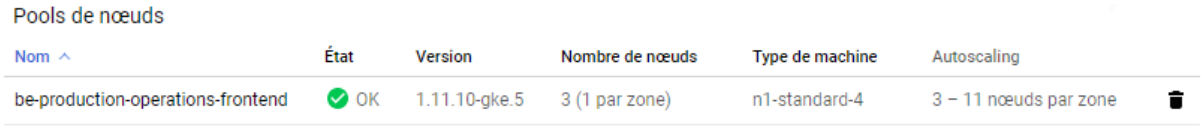

Figure IV.24. Cluster support

n1-standard-4 : Type de machine standard disposant de 4 processeurs virtuels et de 15 Go de mémoire

Figure IV.25. Liste des nœuds de cluster support

Les instances maîtres de cluster GKE sont automatiquement mises à niveau pour exécuter les nouvelles versions de Kubernetes au fur et à mesure qu'elles deviennent stables.

Nous pouvons visualiser la répartition des nœuds sur des zones différentes ce qui assure la haute disponibilité du cluster en cas de sinistre, nous avons mis jusqu'à 11 nœuds pour l'autoscaling par zone (Google offre jusqu'aux 1000 nœuds par zone pour faire l'autoscaling).

L'autoscaler de cluster mesure l'utilisation de chaque nœud par rapport à la demande totale de capacité du pool de nœuds. Si aucun pod n'a été planifié sur un noeud pendant une période définie et si tous les pods s'exécutant sur ce nœud peuvent être planifiés sur d'autres nœuds du pool, l'autoscaler déplace les pods et supprime le nœud.

Region : Les pojets sont localiser dans la region europe-west1 (Belgique)

IV.2.2. Mise en place de VPC

Après avoir préparé nos grappes Kubernetes, avec ses différents configurations, nous devons leur attribuer un VPC qui peut configurer automatiquement la topologie virtuelle des clusters en définissant les plages de préfixes pour les sous-réseaux ainsi que les règles de pare-feu, il consiste configurer la topologie de réseaux des clusters et les faire communiquer les en privé à l'échelle regional.

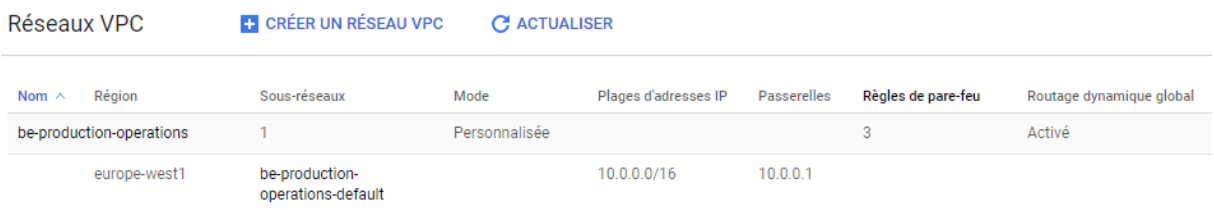

Figure IV.26. Réseaux VPC

IV.2.3. Mise en place des zones DNS

Pour chaque cluster nous avons configurer deux zones de DNS :

- **Interne :** pour assurer la communiquation en interne entre les différents clusters à travers le résaux VPC que nous avons configuré.
- **Externe :** pour assuer la communiquation en externe (internet) à traver l'équilibreur de charge de cluster GKE.

Nous pouvons remarquer que le cluster support avoir l'accés pour toutes les DNS privées des autre clusters.

Cloud DNS CRÉER UNE ZONE

Nom de zone	Nom DNS	DNSSEC	Description	Saisissez	Utilisées par	Appairage DNS
be-no-production-applications-private	nopapp.		European applications noprod private zone	privé	1 réseau	Activé
be-production-applications-private	prdapp.		European applications production private zone	privé	1 réseau	Activé
be-production-operations-private	prdops.		Operations production private zone	privé	1 réseau	Désactivé
be-production-operations-public	prdops.timq.fr.	Off	Operations production public zone	public		

Figure IV.27. Liste des zones DNS de cluster support

IV.2.4. Mise en place de serveur d'intégration 'Jenkins'

Dans notre cas, le déploiement de Jenkins va être au niveau de cluster support (prdops). Dans cette partie, nous détaillons la spécification et l'automatisation de la mise en place de serveur Jenkins, nous avons utilisé la charte officiel de Helm (stable/jenkins).

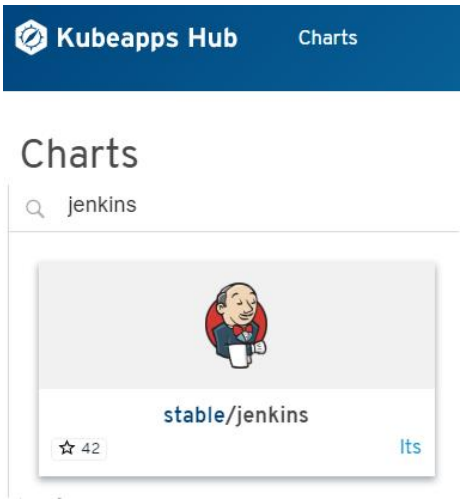

Figure IV.28. Charte Jenkins

IV.2.4.1. Installation de serveur Tiller

En premier lieu, nous devons déployé le serveur Tiller dans l'espace de nom kube-system de notre cluster afin qu'il puisse éxecuter des déploiements au dessus.

```
souhail@souhail:~$ kubectl get deployments -n kube-system
NAME                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
event-exporter-v0.2.4  1         1         1            1           66d
fluentd-gcp-scaler     1         1         1            1           141d
heapster-v1.6.0-beta.1 1         1         1            1           141d
kube-dns               2         2         2            2           141d
kube-dns-autoscaler    1         1         1            1           141d
l7-default-backend     1         1         1            1           141d
metrics-server-v0.2.1  1         1         1            1           141d
tiller-deploy          1         1         1            1           66d
souhail@souhail:~$
```

Figure IV.29. Déploiement de serveur Tiller

Au fur et à la mésure, nous avons crée un service account et nous avons lui attribué le role cluster-admin pour que le serveur Tiller avoir la permission de créer, modifier et supprimer des objets Kubernetes (pods, deploiement..)

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl describe clusterrolebindings | grep terraform-tiller
Name:         terraform-tiller
  ServiceAccount  terraform-tiller  kube-system
```

Figure IV.30. ClusterRoleBinding de serveur Tiller

IV.2.4.2. Personnalisation de la charte Jenkins

Jnekins Charte contient un fichier paramètres configurable que nous pouvons personnaliser tout en utilisant le plugin Helm de terraform.

A des besoins de notre client nous avons presonnalisé l'implémentation de la charte jenkins:

Spécification de l'image jenkins slave :

Nous avons implémenter l'image 'adriagalin/jenkins-jnlp-slave' pour faire créer les pods jenkins slave, cette image est une instance de l'image officiel 'jenkins/jnlp-slave' qui contient les binaires de Docker afin qu'on arrive à faire le build des application conteneurisé.

Installation des plugins :

- Git Plugin : permet à Jenkins d'interagir avec le serveur de gestion de version GIT (Github)
- Pipeline Plugin : permet à Jenkins de créer des « Job » en se basant sur l'utilisation des scripts au lieu de l'interface graphique

- Kubernetes Plugin : permet d'utiliser des comptes de service Kubernetes pour l'authentification, et de créer des configurations d'exécuteur (agent). Le plugin crée un pod lorsqu'un exécuteur est requis et le détruit lorsqu'une tâche se termine.

Claim un volume persistant :

Par défaut, les fichiers de disque d'un conteneur sont éphémères. Cela signifie que lorsque le conteneur Jenkins échoue, détruit ou bien lors d'une mise à niveau, les données du répertoire JENKINS_HOME sont définitivement perdues .

Il serait souhaitable de conserver le répertoire JENKINS_HOME. Cela peut être réalisé avec les volumes Kubernetes .

Ces données sont stockées sur un disque persistant géré par le cluster GKE et persisteront lors des redémarrages de Jenkins.

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl describe pv
Name:            prdops-jenkins-home
Labels:          failure-domain.beta.kubernetes.io/region=europe-west1
                 failure-domain.beta.kubernetes.io/zone=europe-west1-b
Annotations:     pv.kubernetes.io/bound-by-controller: yes
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    ssd
Status:          Bound
Claim:           default/prdops-jenkins-home-claim
Reclaim Policy:  Retain
Access Modes:    RWO
Capacity:        8Gi
Node Affinity:   <none>
Message:
Source:
    Type:       GCEPersistentDisk (a Persistent Disk resource in Google Compute
Engine)
    PDName:     prdops-jenkins-home-disk
    FSType:
    Partition:  0
    ReadOnly:   false
Events:         <none>
```

Figure IV.31. Description de volume persistant

Un volume est monté au-dessus du système de fichiers du conteneur, les données sont donc toujours lues de la même manière qu'auparavant, sauf que des modifications seront apportées au volume. Comme le volume est durable après le redémarrage du conteneur (redémarrage de routine ou en cas d'échec), les données sont conservées.

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl describe pvc
Name:          prdops-jenkins-home-claim
Namespace:     default
StorageClass:  ssd
Status:        Bound
Volume:        prdops-jenkins-home
Labels:        <none>
Annotations:   pv.kubernetes.io/bind-completed: yes
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:      8Gi
Access Modes:  RWO
Mounted By:    jenkins-6f9d8d7687-xb4w9
Events:        <none>
```

Figure IV.32. Description de volume persistant claim

IV.2.4.3. Implémentation de serveur Jenkins

La figure ci-dessous montre ce qui s'est passé. Helm a créé un déploiement et quelques services sur notre cluster k8s. Le déploiement définit un état souhaité pour le pod contenant le maître Jenkins. Celui est exposé en externe (accessible depuis n'importe quel point d'internet) à l'aide de l'équilibreur de charge du notre cluster, nous pourrons toujours accéder au maître Jenkins via l'IP de l'interface sur le port 8080. Cependant, le service d'agent Jenkins est exposé à l' IP de cluster, ce qui signifie qu'il n'est accessible que depuis le cluster.

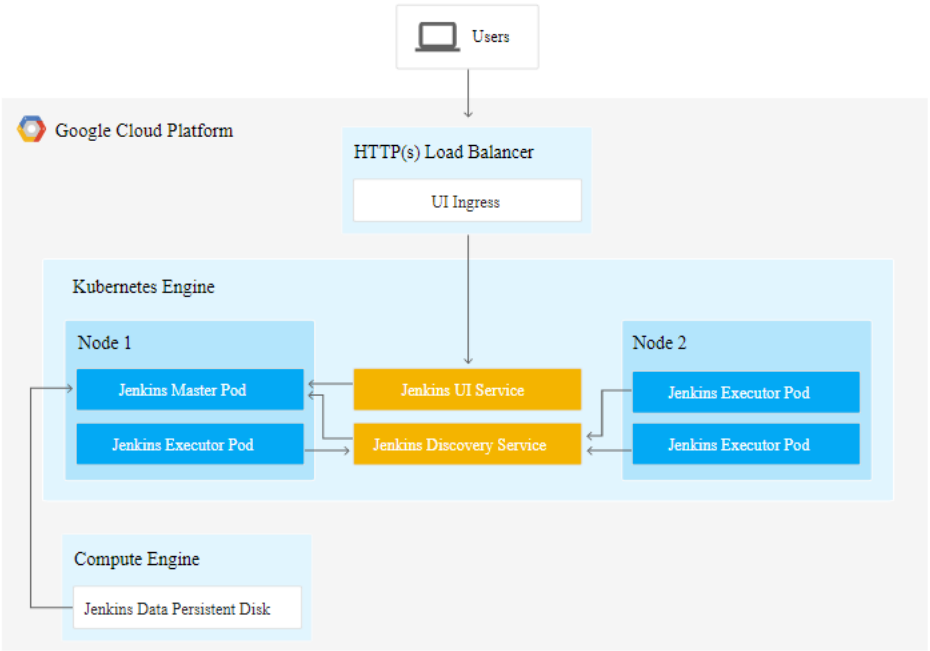

Figure IV.33. Architecture Jenkins

Le maître Jenkins est déployé avec un nombre d'instances dupliquées égal à 1. Cela permet de s'assurer qu'un seul maître Jenkins est en cours d'exécution dans le cluster à tout moment. Si

le pod maître Jenkins meurt ou si le nœud sur lequel il est exécuté est arrêté, Kubernetes redémarre le pod ailleurs dans le cluster.

IV.3. Livraison continue

Dans cette section, nous détaillons la configuration du serveur d'intégration continue Jenkins. Pour tester cette chaine nous avons utilisé une application nommée « macha », c'est une application basée sur une architecture de conteneurisation. Par la suite, nous exposons les différentes phases pour réussir à mettre en œuvre notre pipeline de livraison continue.

IV.3.1. Lancer le pipeline de build

D'abord, nous avons intégrer le projet GitHub au moment de création de pipeline, ce projet contient les fichiers Docker-compose et un fichier Jenkinsfile qui décrit toutes les phases de build de l'application.

La Figure 26 comporte une imprime écran sur les jobs Jenkins contenant un script pipeline définissant la préparation, le build et le stockage de l'image Docker comme étant les étapes d'un pipeline appliqué sur l'application.

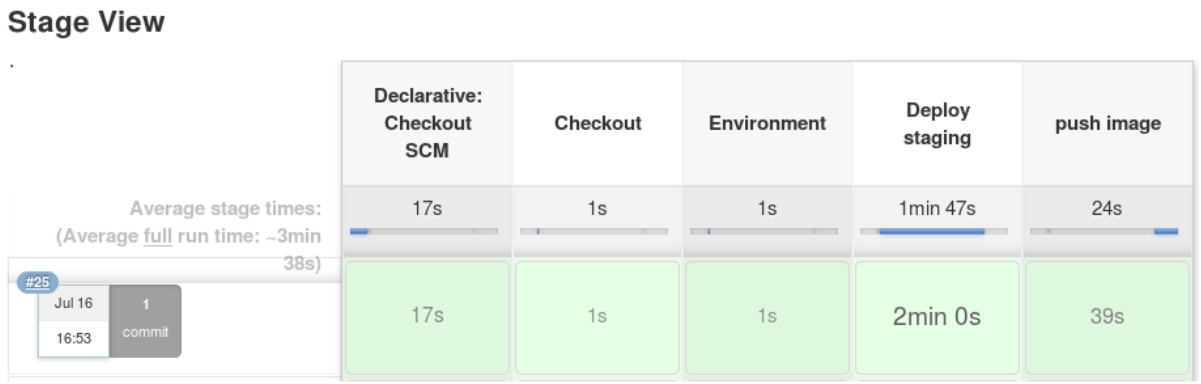

Figure IV.34. Pipeline de build

Toutes les images créée vont être stockées dans notre registre privée GCR sous le référentiel de build. Le registre est une application côté serveur hautement évolutive permet de stocker et de distribuer des images Docker. La Figure 27 ci-dessous représente le registre privé de notre application conteneurisé.

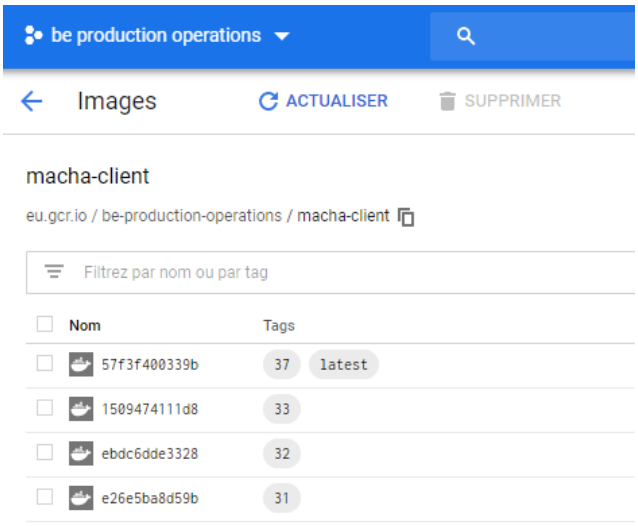

Figure IV.35. Registre des images Docker

La figure ci-dessous décrit le flux de travail de Jenkins gérer par Kubernetes du moment de création de pod jusqu'à ce que la construction soit terminée. Ensuite, l'esclave publiera les résultats de construction existants sur le maître et se terminera.

```
Events:
  Type    Reason     Age   From
         Message
  ----    ------     ----  ----
         -------
  Normal  Scheduled  19s   default-scheduler
         Successfully assigned default/jenkins-agent-2kzx0 to gke-be-production-
op-be-production-op-3c99535d-9b0p
  Normal  Pulling    18s   kubelet, gke-be-production-op-be-production-op-3c9953
5d-9b0p  pulling image "adriagalin/jenkins-jnlp-slave:latest"
  Normal  Pulled     16s   kubelet, gke-be-production-op-be-production-op-3c9953
5d-9b0p  Successfully pulled image "adriagalin/jenkins-jnlp-slave:latest"
  Normal  Created    16s   kubelet, gke-be-production-op-be-production-op-3c9953
5d-9b0p  Created container
  Normal  Started    16s   kubelet, gke-be-production-op-be-production-op-3c9953
5d-9b0p  Started container
souhail@souhail:~/devops-infra-base/terraform$ kubectl get pods
NAME                       READY   STATUS    RESTARTS   AGE
jenkins-6f9d8d7687-xb4w9   1/1     Running   0          5d
jenkins-agent-2kzx0        1/1     Running   0          39s
souhail@souhail:~/devops-infra-base/terraform$
```

Figure IV.36. Description de création de pod jenkins slave

IV.3.2. Automatisation de déploiement

Une fois notre application conteneurisé est stocker à notre registre. Nous pouvons automatisé son déploiement sur notre plate-forme, cela nécessite la création des différentes ressources et des objets Kubernetes.

Nous allons decrit de ce qui suit le résultats de différentes étapes de déploiement.

D'abord, nous avons commencé par l'implémentation d'un namespace, ce dernier est un moyen de séparer logiquement un cluster physique en plusieurs clusters virtuels.

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl get namespaces | grep macha-namespace
macha-namespace    Active    53d
```

Figure IV.37. Macha namespace

Ensuite, nous avons mis en place l'objet deployment, celui qui s'occuper de tirage de l'image docker stocker chez GCR, de plus, il fournit des mises à jour déclaratives pour les pods, comme la déclaration d'un nouvel état, la mise en pause et le nettoyage des déploiements.

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl get deployments -o wide -n macha-namespace
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS     IMAGES
macha-prdapp   2         2         2            2           13m   macha-client   eu.gcr.io/be-production-operations/macha-client
```

Figure IV.38. Macha deployment

Pour assurer la haut disponibilité de l'application, nous avons répliqué deux instances de l'application, chacune sur un pod différent. Chaque pod encapsule plusieurs conteneurs nécessaires au fonctionnement de l'application Macha.

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl get pods -n macha-namespace
NAME                            READY   STATUS    RESTARTS   AGE
macha-prdapp-76c665858c-2mg7j   1/1     Running   0          12m
macha-prdapp-76c665858c-zxpps   1/1     Running   0          12m
```

Figure IV.39. Macha pods

C'est le service qui va se charger de router les requêtes vers les pods qu'il gère. Contrairement aux pods, un service est persistant et représente le service exposé au reste du cluster. Il permet d'être découvert via le mécanisme de service Discovery fourni par Kubernetes, nous avons lui fournit une addresse IP à travers l'équilibreur de charge exposé au port 80.

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl get svc -o wide -n macha-namespace
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP    PORT(S)        AGE   SELECTOR
macha-prdapp   LoadBalancer   10.59.248.154   34.77.65.202   80:31667/TCP   11m   app=macha-prdapp
```

Figure IV.40. Macha service

Par suite, nous avons configuré un Ingress pour lier le service à un URL accessibles à partir de l'extérieur, assurer un partage de charge de trafic à travers le service DNS.

```
souhail@souhail:~/devops-infra-base/terraform$ kubectl get ingress -o wide -n macha-namespace
NAME            HOSTS                          ADDRESS         PORTS   AGE
macha-ingress   macha-client.prdapp.tlmq.fr    35.244.148.70   80      11m
```

Figure IV.41. Macha ingress

Enfin, nous avons notre application déployé automatiquement dans un Kubernetes cluster hébergée chez Google Cloud.

Figure IV.42. Macha application

Conclusion

Dans ce chapitre, nous avons présenté la phase de réalisation de_notre projet. Nous avons décrit l'environnement du travail et l'architecture déployée. Nous avons aussi décrit les étapes de mise en place de notre plate-forme de conteneurisation et de livraison continue. Enfin, nous avons fini par présenter l'automatisation de déploiement de l'un des applications conteneurisé Macha.

Conclusion Générale

Arrivé au terme de ce rapport, il est important de rappeler que l'objectif de ce travail était de la fourniture automatique d'une plateforme de livraison continue basée sur une architecture de conteneurisation hébergée chez le fournisseur Google Cloud.

Pour atteindre cet objectif, nous avons tout d'abord étudié le concept du Cloud computing, la méthodologie DevOps et les solutions existantes dans ce domaine. Ensuite, nous avons déterminé les différents besoins relatifs à notre solution pour aboutir à la partie conception. Puis, nous avons enchaîné avec une étude technique dans laquelle nous avons présenté l'environnement de travail pour la réalisation du projet. Enfin, nous avons abordé la partie réalisation à travers laquelle nous avons décrit les différentes fonctionnalités de notre solution.

Pour conclure, notre modèle a bien résolu les différentes limites présentées par les solutions existantes et nous a donc permis d'atteindre les objectifs fixés. En effet, notre solution garantit l'automatisation de l'approvisionnement des ressources à travers le Cloud afin de minimiser l'intervention humaine dans les opérations IT, la réduction de l'intervention des différentes parties prenantes : développeurs, intégrateurs et l'accélération du processus d'intégration et de livraison. Durant ce projet, nous avons également été confrontés à de nombreuses dicultés techniques, surtout liées à la compatibilité des technologies et leurs versions. Mais, l'interaction avec la communauté à travers en autre la plateforme slack, des seances de chats et de déplacement chez le client nous a permis de discuter les problèmes rencontrés et les solutions adoptées et implémentées.

Nétographie

[1] : https://www.mikadolabs.com/ [consulté le 18 mars 2019]

[2] : https://fr.ryte.com/wiki/Mod%C3%A8le_en_spirale [consulté le 21 mars 2019]

[3] : https://medium.com/edureka/ci-cd-pipeline-5508227b19ca [consulté le 25 mars 2019]

[4] : https://www.lemagit.fr/definition/Conteneur [consulté le 30 mars 2019]

[5] : https://www.hpe.com/fr/fr/what-is/container-orchestration.html [consulté le 15 avril 2019]

[6] : https://www.redhat.com/fr/topics/automation/whats-it-automation [consulté le 15 avril 2019]

[7] : https://www.lebigdata.fr/docker-definition [consulté le 27 avril 2019]

[8] : https://hackernoon.com/kubernetes-vs-docker-swarm-a-comprehensive-comparison-73058543771e [consulté le 12 may 2019]

[9] : https://kubernetes.io/docs/concepts/ [consulté le 12 may 2019]

[10] : https://www.ionos.com/digitalguide/server/know-how/caas-container-as-a-service-service-comparison/ [consulté le 20 juin 2019]

[11] : https://www.blazemeter.com/blog/how-to-setup-scalable-jenkins-on-top-of-a-kubernetes-cluster/ [consulté le 30 juin 2019]

[12] : Cours UML 3A Esprit [consulté le 27 avril 2019]

Résumé

Le présent rapport est le résultat d'un travail réalisé dans le cadre d'un projet de fin d'études au sein de la société Mikado Labs pour une durée de six mois sous l'encadrement de Mr. Julien Beaucourt, Manager technique de la société et Raphael Monroueau l'expert Devops.

Afin d'assurer la bonne gestion des cycles de vie des applications et améliorer le processus de la livraison des projets, Mikado Labs a opté pour l'implémentation d'une plateforme de livraison continue, ainsi que la mise en place de son propre environnement de conteneurisation afin d'automatiser le processus de la livraison du projet et améliorer le « Time to Market ».

C'est dans cette optique et en guise de projet de fin d'études que nous avons été appelés pour implémenter cette solution. La mise en œuvre de ce projet met l'accent sur l'utilisation des nouvelles technologies pour résoudre des problématiques diverses.

Abstract

The present document is the result of a six-month end-of-studies internship at Mikado Labs. The internship was accomplished under the supervision of Mr. Julien Beaucourt, the technical manager. In order to ensure the good management of applications lifecycle and enhance the delivery process, Mikado Labs chose to implement a pipeline for Continous delivery, as well as to setup a conteneurized envirement to automate the delivery process and boost the « time to market ».

Within this context, we brought this project to light using several technologies to solve various problems.