
gestion des accents (PDF)

RAPPORT DE PROJET DE FIN D'ÉTUDES

Présenté en vue de l'obtention du
MASTER CO-CONSTRUIT EN SÉCURITÉ DES RÉSEAUX ET
COMMUNICATIONS EMBARQUÉES

Spécialité : Sécurité des Réseaux et Communications Embarquées

Étude et implémentation d'un pipeline CI/CD d'une application conteneurisée orchestrée par GKE

Par
NOUR TEBER

Réalisé au sein d'Oyez-T



Soutenu publiquement le 21 septembre 2022 devant le jury composé de :

Présidente : Mme Maha CHERIF, Enseignante, ISTIC

Rapporteur : M Hichem ARFA, Enseignant, ISTIC

Encadrant professionnel : M Mohamed Amine GHERIBI, Consultant DevOps,
Oyez-T

Encadrant académique : Mme Nejla ESSADDI, Enseignante, ISTIC

Lu et Approuvé
Le 25-09-2022
Mr Hichem Arfa

Année Universitaire : 2021-2022

RAPPORT DE PROJET DE FIN D'ETUDES

Présenté en vue de l'obtention du
MASTER CO-CONSTRUIT EN SÉCURITÉ DES RÉSEAUX ET
COMMUNICATIONS EMBARQUÉES

Spécialité : Sécurité des Réseaux et Communications Embarqués

Étude et implémentation d'un pipeline CI/CD d'une application conteneurisée orchestrée par GKE

Par
NOUR TEBER

Réalisé au sein d'Oyez-T



Autorisation de dépôt du rapport de Projet de Fin d'Etudes :

Encadrant professionnel :
Mohamed Amine GHE-
RIBI

Encadrant académique :Mme Nejla
ESSADDI

Le :

Le :

Signature :

Signature :

Dédicaces

Je dédie cet ouvrage

A mon cher père Youssef, qui m'a donné la main d'aide toutes les années précédentes.

A ma chère mère Latifa, qui m'a encouragé durant les années de mes études.

A mes frères Mohamed et Ramzi, qui ont partagés avec moi tous les beaux moments lors de la réalisation de ce travail.

A mes chers amis, qui m'ont aidé et supporté dans les moments difficiles.

Remerciements

C'est avec le plus grand honneur que nous avons réservé cette page en signe de gratitude et de reconnaissance à tous ceux qui nous ont aidés de près ou de loin à l'aboutissement de ce travail dans les meilleures conditions, sans leur aide inestimable, ce projet n'aurait jamais été mené à son terme. Nos remerciements s'adressent particulièrement à :

- ⌘ Notre superviseur professionnel M. GHERIBI Mohamed Amine, que nous voudrions exprimer notre gratitude pour l'aide qu'il nous a apporté durant toutes les phases de ce projet.
- ⌘ Notre encadrant académique , Mme. Nejla ESSADDI, pour nous a guidé et nous a aidé dans la rédaction de notre rapport.
- ⌘ Un grand merci au membre du jury qui ont acceptés à participer à l'évaluation de notre projet, nous souhaitons qu'il sera à la hauteur de vos attentes.

Table des matières

Dédicaces	i
Remerciements	ii
Introduction Générale	2
1 Cadre général du projet	4
1.1 Introduction	4
1.2 Présentation de l'organisme d'accueil	4
1.2.1 Services et Secteurs d'activité	5
1.2.2 Produits et solutions	5
1.2.3 Les partenaires d'Oyez-T	5
1.2.4 Les clients d'Oyez-T	6
1.3 Présentation du projet	7
1.3.1 Étude de l'existant	7
1.3.2 Étude comparative Agile vs DevOps	7
1.3.3 Critique de l'existant	10
1.3.4 Solution Proposé	10
1.4 Conclusion	11
2 Étude Préalable et Architecture générale proposée	12
2.1 Introduction	12
2.2 Étude Préalable	12
2.2.1 DevOps	12
2.2.2 Intégration Continue / Déploiement Continue	14
2.2.3 Pipeline CI/CD	15
2.3 Architecture générale proposée	16
2.3.1 Les outils à utiliser	17
2.3.2 Virtualisation	19
2.3.3 Les orchestrateurs de conteneurs	26
2.4 Conclusion	31
3 Conteneurisation	32
3.1 Introduction	32
3.2 Architecture de conteneurisation proposée	32
3.3 Environnement du travail	33
3.3.1 Les objets de docker	33
3.3.2 Docker-Compose	34
3.3.3 Registre Docker	35

3.3.4	Les serveurs web	36
3.4	Conteneurisation du projet	36
3.4.1	Mise en place de l'environnement de travail	37
3.4.2	Conteneurisation des micro-services	38
3.4.3	Docker-compose	45
3.4.4	Registre de conteneurs	51
3.5	Conclusion	54
4	Déploiement des micro-services sur GKE	55
4.1	Introduction	55
4.2	Architecture de déploiement proposée	55
4.3	Environnement du travail	56
4.3.1	Google Cloud Plateforme	57
4.3.2	Bitbucket	59
4.4	Etape de déploiement des micro-services	60
4.4.1	Provisionnement de l'environnement de travail	61
4.4.2	Pipeline CI/CD	64
4.4.3	Déploiement de l'application sur le cluster GKE	70
4.4.4	Présentation des interfaces	73
4.5	Conclusion	75
Conclusion Générale	76	
Webographie	77	

Table des figures

1.1	Oyez-T	4
1.2	Elasticsearch	6
1.3	Apollo	6
1.4	Les clients d'Oyez-T	6
1.5	Architecture actuelle	7
1.6	DevOps vs Agile	8
1.7	Agile	8
1.8	DevOps	9
2.1	Les pratiques DevOps	13
2.2	L'approche CI/CD	14
2.3	Architecture Générale proposée	16
2.4	SVN	18
2.5	Git	18
2.6	Architectures des machines virtuelles	20
2.7	Modèles de services de Cloud	22
2.8	Les types de cloud	23
2.9	Amazon Web Services (AWS)	24
2.10	Google Cloud Plateform	24
2.11	Docker	26
2.12	Les composants de Docker	27
2.13	Kubernetes	28
2.14	Cluster kubernetes	30
3.1	Architecture de conteneurisation proposée	33
3.2	Les objets Docker	34
3.3	Docker-Compose	35
3.4	Harbor	35
3.5	Nginx	36
3.6	Script d'installation Docker	37
3.7	Exécution de script d'installation	37
3.8	Statut Docker	38
3.9	Script Docker-Compose	38
3.10	Installation Docker-Compose	38
3.11	Dockerfile 'scope_front'	39
3.12	Docker image 'scope_front'	40
3.13	Conteneur 'scope_front'	40
3.14	L'interface 'scope_front'	41
3.15	Dockerfile 'scope_api'	41

3.16 Docker image 'scope_api'	42
3.17 Conteneur 'scope_api'	42
3.18 L'interface 'scope_api'	42
3.19 Installation du service 'MongoDB'	43
3.20 Statut 'MongoDB'	43
3.21 Ajout user Mongo	43
3.22 Dockerfile et .env.dev	44
3.23 Docker-compose.yml	44
3.24 L'interface 'scope_backoffice'	45
3.25 Configuration docker-compose.yml	46
3.26 Déploiement des conteneurs	47
3.27 Statut 'Nginx'	47
3.28 Configuration Nginx 'scope_front'	48
3.29 Interface 'local.scope-front'	48
3.30 Configuration Nginx 'scope_api'	49
3.31 Interface 'local.scope-api'	49
3.32 Configuration Nginx 'scope_backoffice'	49
3.33 Interface login du micro-service 'scope_backoffice'	50
3.34 Interface 'local.scope-backoffice'	50
3.35 Interface 'Portainer'	51
3.36 Fichier de configuration 'harbor.yml'	52
3.37 Installation 'Harbor'	52
3.38 Hébergement 'scope_front'	53
3.39 Hébergement 'scope_api'	53
3.40 Hébergement 'scope_backoffice'	54
4.1 Architecture de déploiement proposée	56
4.2 Les services de 'GCP'	57
4.3 Bitbucket	60
4.4 Bitbucket pipeline	60
4.5 Création d'un nouveau projet	61
4.6 Création d'un compte service	62
4.7 Création du réseau VPC	62
4.8 Création du registre docker	63
4.9 Création cluster GKE 'scope-recette'	63
4.10 Création des nœuds	64
4.11 Pipeline 'scope_front'	65
4.12 Répertoire des variables du 'Bitbucket'	66
4.13 Script 'config.sh'	66
4.14 Script 'config.sh'	67
4.15 Arborescence Helm chart 'scope_front'	68
4.16 Instruction de déploiement 'scope_front'	68
4.17 Pipeline 'scope_front'	69
4.18 Pipeline 'scope_api'	69
4.19 Pipeline 'scope_backoffice'	70
4.20 Le registre cloud du micro-service 'scope_front'	70
4.21 Le registre cloud dédié au micro-service 'scope_api'	71
4.22 Allocation des images docker du micro-service 'scope_backoffice'	71

4.23 Création des pods	72
4.24 Les services créés	72
4.25 les ingress de chaque micro-service.	73
4.26 Les ingress de chaque microservice.	73
4.27 Interface du micro-service ‘scope_front’	74
4.28 Interface de login ‘scope_backoffice’	74
4.29 Interface graphique ‘scope_backoffice’	75

Liste des tableaux

1.1	Comparaison Agil / Devops	9
2.1	Comparaison entre SVN et Git	19
2.2	Comparaison entre Virtualisation et Conteneurisation	21
2.3	Comparaison entre AWS et GCP	25

Abréviations

- ❖ **DEV** : Développement.
- ❖ **OPS** : Opérationnel.
- ❖ **K8S** : Kubernetes.
- ❖ **GCP** : Google Cloud Platform.
- ❖ **GKE** : Google Kubernetes Engine.
- ❖ **IAM** : Identity and Access Management.
- ❖ **VPC** : Virtual Private Cloud.
- ❖ **CI** : Intégration Continue.
- ❖ **CD** : Déploiement Continu.
- ❖ **VM** : Machine Virtuelle.
- ❖ **Qualif** : Environnement de test.
- ❖ **SVN** : Subversion

Introduction Générale

La productivité dans de nombreuses entreprises est en baisse, principalement en raison d'un manque de communication entre les équipes, affectant leurs performances, faisant ainsi perdre du temps et de l'argent.

De plus, la génération de nouvelles solutions suit la méthode de travail classique, qui se présente comme un processus continu bien défini comprenant des phases de planification, de développement, de test et de livraison. Ces approches s'appuient sur deux équipes différentes, une équipe de développement intéressée par la conception et l'implémentation du code et effectuant des tests locaux avant de le remettre à l'équipe d'exploitation responsable du déploiement.

Cela implique des conflits car chaque équipe a des objectifs différents, voire opposés, de sorte que chacun ne se soucie que de son propre bastion, ce qui entraîne des livraisons ratées et des clients mécontents.

Et après de nombreuses recherches, le concept de DevOps est né, c'est une vision du travail née de l'évolution agile, qui repose sur la synergie de la partie opérationnelle et de la partie production.

Cette culture affecte toutes les phases du cycle de vie du logiciel. En rassemblant les compétences, les processus et les outils de l'ensemble de l'organisation informatique et d'ingénierie. Il joue un rôle très important dans le développement de l'entreprise, l'amélioration des délais de production (Time-To-Market) et la garantie de la satisfaction client.

Dans ces circonstances, nous effectuerons notre stage de fin d'étude au sein de l'entreprise "Oyez-T", et notre projet comprend la construction d'un pipeline d'intégration/déploiement continu d'applications conteneurisées orchestré par GKE. Son objectif est d'automatiser le déploiement des applications afin de réduire le temps et les coûts d'exécution.

Ce rapport est organisé autour des quatre chapitres suivants :

- ❖ Le premier chapitre, intitulé « Cadre général du projet », est consacré à présenter l'organisme d'accueil et à mettre en place notre projet dans son contexte global en identifiant la solution à adopter.
- ❖ Le deuxième chapitre intitulé « Étude Préalable et Architecture générale proposée » a pour mission de démontrer le concept DevOps voire l'approche intégration continue/déploiement continu, en illustrant l'architecture globale proposée et les outils

adoptés.

- ❖ Le troisième chapitre intitulé « Conteneurisation » répond au premier objectif de notre projet, où nous décrirons l'architecture détaillée de cette phase, et les outils nécessaires à son exécution.
- ❖ Le quatrième chapitre intitulé « Déploiement des micro-services sur GKE » est consacré à l'utilisation du pipeline CI/CD pour automatiser le déploiement d'une application sur une plateforme cloud.

Chapitre 1

Cadre général du projet

1.1 Introduction

Dans ce premier chapitre, nous présenterons le projet dans son cadre général en présentant l'organisme d'accueil où nous effectuons notre stage de fin d'études, ses services et ses clients.

De plus, nous montrerons l'architecture actuelle approuvée à la production et illustrerons ses problèmes afin de présenter notre solution proposée.

Et à la fin nous vous présenterons le diagramme de Gantt qui reprend les différentes tâches à réaliser.

1.2 Présentation de l'organisme d'accueil

Oyez est une agence experte en commerce connecté, indépendante fondée en 2006 par Olivier Nachba, Henri Danzin et Marie-Agnès Danzin.

Elle combine la maîtrise des parcours client connectés, des technologies et des processus pour accélérer la transformation digitale des marques et des distributeurs au service de l'expérience client personnalisée.

Forte de ses 70 collaborateurs répartis entre Paris et Tunis, l'agence déploie des solutions innovantes et créatives fondées sur l'humain, l'innovation et les données auprès d'acteurs leaders du retail, de l'automobile, du luxe et des services tels que : BNP Paribas, Carrefour, Clarins, E.Leclerc, Feu Vert, Franprix, Galeries Lafayette, LCL, LVMH, Maison du Monde, Monoprix, Saint Laurent, etc.[?]



FIGURE 1.1 – Oyez-T

OYEZ Tunis est à la fois un pôle de développement d'excellence, un centre de services et de formation. En fonction des problématiques et des besoins clients, nous offrons différents modes d'organisation allant de l'équipe dédiée à la prestation on-demand avec une grande flexibilité.

Cette diversité permet de co-construire le mode de fonctionnement optimal avec le client. En fonctionnant main dans la main avec les équipes Parisiennes et en se concentrant sur les technologies Cloud et le traitement de la donnée, Oyez-T propose une alternative à forte valeur ajoutée aux modèles classiques d'Offshoring. Avec des compétences dans l'ensemble des domaines critiques des projets digitaux orientés commerce. [?]

1.2.1 Services et Secteurs d'activité

Oyez-T offre la flexibilité nécessaire pour remédier aux problèmes. Il apporte les compétences nécessaires en mode Offshore, dédiées à la production des projets.

Les domaines d'activités d'Oyez sont :

- ★ Commerce connecté.
- ★ Services Marketing.
- ★ Communication Digitale.
- ★ Digital retail.
- ★ Machine learning.
- ★ Devops.
- ★ Cloud.
- ★ E-commerce.
- ★ M-commerce.
- ★ Digital in-store[?]

1.2.2 Produits et solutions

Oyez propose une gamme de produits très diversifiée, parmi ses produits nous citons :

- ❖ **Solutions Retail(Tech)** : Dans un environnement où l'accélération la transformation retail devient un sujet stratégique pour les directions digitales, oyez développe des outils et intègre des technologies de rupture leur permettant de retrouver la souplesse et évolutivité que les projets omni-canaux doivent devenir de véritables vecteurs de développement du business.
- ❖ **Solutions Digital Shopper** : L'extrême exigence des nouveaux clients multi connectés impose de nouvelles règles aux marques. Leurs services digitaux doivent désormais être à la hauteur de leurs promesses en terme de valeur ajoutée. Oyez concave les parcours, les applications, les interfaces, les contenus et les services digitaux valorisant l'expérience de marque tout au long des parcours d'achat.[?]

1.2.3 Les partenaires d'Oyez-T

Le groupe d'Oyez-T a plusieurs partenaires, dont : Elasticsearch, Apollo, etc...

Elasticsearch

Elasticsearch est un moteur de recherche et d'analyse distribué gratuit et ouvert pour tout type de données, y compris les données textuelles, numériques, géospatiales, structurées et non structurées.



FIGURE 1.2 – Elasticsearch

Apollo

Apollo est une plateforme de prospection efficace qui rassemble toutes les fonctionnalités en un seul endroit. L'extension facile à prendre en main et les intégrations permettent rapidement d'utiliser l'outil de manière efficace. L'outil reste un peu cher.



FIGURE 1.3 – Apollo

1.2.4 Les clients d'Oyez-T

La figure 1.4 illustre les différents clients d'Oyez-T :



FIGURE 1.4 – Les clients d'Oyez-T

1.3 Présentation du projet

Au cours de cette partie nous présenterons le procédé actuel d'Oyez-T adopté dans la construction de ses projets en identifiant les problèmes, ainsi que la solution proposée.

1.3.1 Étude de l'existant

Oyez-T suit une approche classique à la gestion de ses projets, dont une équipe de DEV s'intéresse à la planification et au développement des nouvelles fonctionnalités.

Ensuite une équipe de test (QA) se présente pour exécuter les différents scénarios de test pour extraire une version livrable au client.

Après chaque étape, le code source doit être hébergé dans le référentiel centralisé "SVN".

A ce moment le travail de l'équipe opérationnelle commence par extraire (pulling) le code cité dans l'annuaire et le déployer sur un serveur OVH loué.

À ce moment, le client peut accéder au projet par un url "HTTPS(s)".

La figure 1.5 illustre l'architecture actuelle :

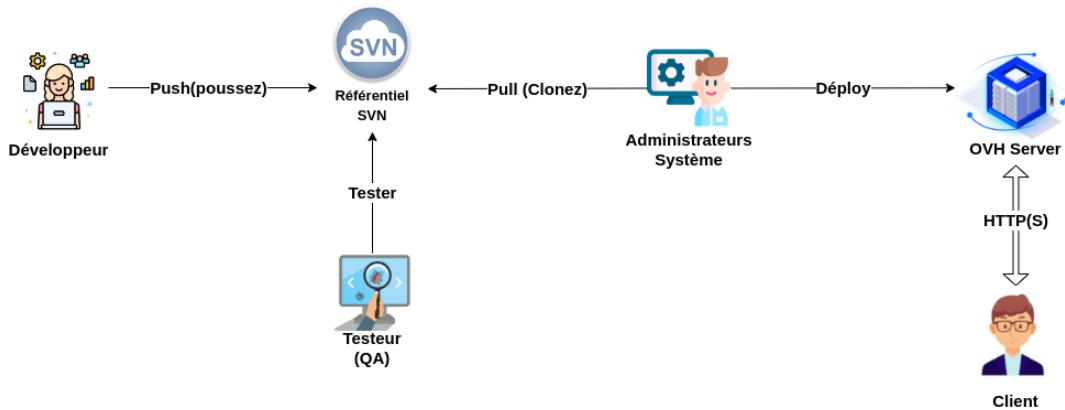


FIGURE 1.5 – Architecture actuelle

De plus Oyez-T utilise la méthodologie Agile précisément le framework Scrum à la gestion de ses projets en devisant le développement de ses solutions à des sprints bien détaillés.

En effet, le processus de code sera incrémenté après chaque sprint qui est déjà approuvé par le client.

1.3.2 Étude comparative Agile vs DevOps

Dans cette partie nous identifiant les deux principales méthodologies de gestion des projets : "Agile" et "DevOps" en dégagent les principales différences entre eux.

La figure 1.6 présente la différence entre ces approches au niveau de production :

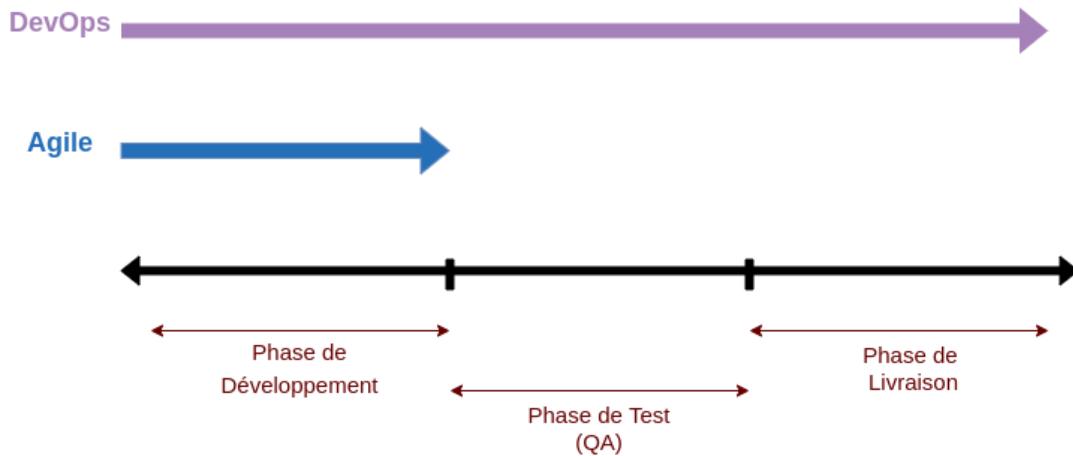


FIGURE 1.6 – DevOps vs Agile

Agile

La méthodologie Agile implique une itération continue du développement et des tests dans le processus SDLC. Cette méthode de développement logiciel met l'accent sur le développement itératif, incrémental et évolutif. Le processus de développement agile divise le produit en petits morceaux et les intègre pour les tests finaux. Il peut être implanté de plusieurs façons, y compris scrum, kanban, scrum, XP, ...[?]

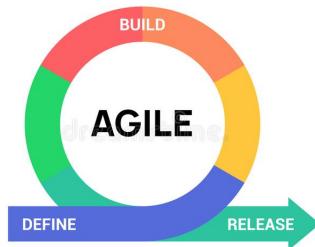


FIGURE 1.7 – Agile

DevOps

DevOps est une méthode de développement logiciel qui se concentre sur la communication, l'intégration et la collaboration entre l'équipe de développement et celle des opérations pour permettre un déploiement rapide des produits et de manière automatisée.[?]

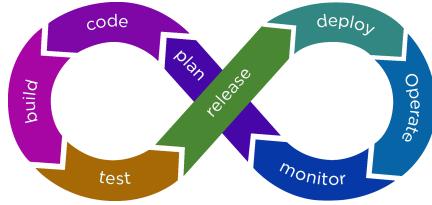


FIGURE 1.8 – DevOps

Le tableau 1.1 résume la différence entre eux selon plusieurs critères :

	Agile	Devops
Définition	Approche itérative se concentre sur la collaboration, les commentaires des clients et les petites versions rapides.	Pratique consistant à réunir les équipes de développement et d'exploitation.
Création	En 2001	En 2007
Objectif	Comble l'écart entre les besoins des clients et les équipes de développement et de test.	Il comble l'écart entre le développement + les tests et l'exploitation.
Tâche	Le processus agile se concentre sur des changements constants.	DevOps se concentre sur des tests et une livraison constante.
Accent	Agile met l'accent sur la méthodologie de développement logiciel pour le développement de logiciels. Lorsque le logiciel est développé et publié, l'équipe agile ne se soucie pas de ce qu'il advient.	DevOps consiste à prendre un logiciel prêt à être publié et à le déployer de manière fiable et sécurisée.
Communication	Scrum est la méthode la plus courante de mise en œuvre du développement logiciel Agile, avec des réunions quotidiennes à effectuer.	Les communications DevOps impliquent des spécifications et des documents de conception.
Automatisation	Agile ne s'intéresse pas par l'automatisation.	L'automatisation est l'objectif principal de DevOps. Il fonctionne sur le principe de maximiser l'efficacité lors du déploiement de logiciels.
Importance	Le développement de logiciels est inhérent à Agile.	L'égalité entre toutes les phases de production.
Avantage	Agile offre un cycle de développement plus court et une meilleure détection des défauts.	DevOps prend en charge le cycle de publication d'Agile.

TABLE 1.1 – Comparaison Agil / Devops

D'après ce tableau, nous constatons que le concept de DevOps est une évolution de la méthodologie Agile avec ses nouvelles pratiques basées sur l'automatisation de la production, et donc la rencontre des équipes DEV avec l'équipe OPS.

1.3.3 Critique de l'existant

D'une façon générale, pour réaliser un projet, Oyez-T a besoin d'une équipe de développement et d'une équipe opérationnelle. La première chargée à développer et améliorer le code avec le moindre coût et temps d'exécution, Tandis que la seconde responsable au déploiement de l'application.

Puisqu'Oyez-T cherche toujours à évoluer et améliorer sa production, l'équipe opérationnelle doit interrompre le déploiement de l'application, à chaque fois, pour que l'équipe de développement puisse apporter des modifications au code, ce qui peut influencer à la stabilité du logiciel, en d'autres termes, cela influencera sa disponibilité.

De plus, les problèmes liés à l'utilisation du référentiel centralisé « SVN » et le codage avec des systèmes d'exploitations différents entraînent des difficultés lors de la gestion des versions de code, ce qui impose une perte de temps dans la phase de développement, voire à la livraison.

Par conséquent, ces problèmes influencent de façon négative le rendement de l'entreprise Oyez-T.

1.3.4 Solution Proposé

Oyez-T est une entreprise qui cherche toujours à mettre en œuvre ses logiciels dans les meilleures conditions possibles tout en respectant les délais de livraison, le coût du projet et la qualité du code.

À cette raison, nous proposons d'adopter le concept de "DevOps" lors de l'automatisation des processus de déploiement d'applications en nous basant sur un outil de gestion de versionning "GIT". Incluant l'intégration, les tests nécessaires et la livraison continu(e)s, et qui garantira une mise en place des applications plus rapide et de bonne qualité. De même, ces applications seront mobiles et exécutables sur n'importe quel environnement(système d'exploitation) utilisant le packaging en images Docker.

D'où l'objectif de notre projet est de mettre en place un pipeline d'intégration continue et de déploiement continu "CI/CD. Notre projet s'articule en deux grandes parties :

- ☒ Conteneuriser l'application avec docker en enregistrant les images docker obtenues dans un registre docker prédéfini.
- ☒ Mettre en place un pipeline d'intégration continue / déploiement continu pour automatiser le déploiement de l'application.
- ☒ Orchestrez les conteneurs d'applications à l'aide de l'outil d'orchestration "kubernetes" pour garantir sa disponibilité.

1.4 Conclusion

Après avoir présenté l'organisme d'accueil et définir la solution proposée de notre projet, c'est le temps d'exposer l'architecture proposée, ainsi les technologies adoptées pour atteindre nos objectifs.

Chapitre 2

Étude Préalable et Architecture générale proposée

2.1 Introduction

Ce chapitre sera divisé en deux grandes parties. Dans une première partie intitulée "Étude Préalable", nous définirons le concept "DevOps" en listant ces pratiques, ensuite, nous détaillerons l'approche intégration continue / déploiement continu (CI/CD).

Et une deuxième partie intitulée "Architecture générale proposée" sera allouée, pour représenter l'architecture générale de notre solution en illustrant les outils et les technologies adoptées.

2.2 Étude Préalable

Nous réservons cette partie pour présenter et définir les termes globaux de notre solution, le « DevOps » et l'approche « CI/CD ».

2.2.1 DevOps

DevOps est un mouvement culturel, un état d'esprit et une philosophie organisationnelle qui mettent l'accent sur la communication, la collaboration et l'intégration entre les équipes de développement "Dev" et les équipes opérationnelles "Ops".

Le concept DevOps insiste sur la sécurité, l'optimisation et l'accélération des valeurs commerciales grâce à la mise en œuvre de nouveaux produits dans un délai plus court tout en garantissant une qualité et une sécurité élevée, ainsi la rapidité de détection et de résolution des bugs.

Par conséquent, cette philosophie organisationnelle repose sur quatre principes :

➤ **Collaboration :**

Facilite la communication mutuelle entre les développeurs et les opérateurs.

➤ **Amélioration continue :**

Innover de nouvelles idées pour améliorer le projet afin de mieux satisfaire le client.

➤ **Le partage :**

Partager le travail (les problèmes rencontrés, les connaissances, etc...), entre tous les membres des équipes.

➤ **L'automatisation des tâches :**

Se résume par l'intégration continue, les tests automatisés et le déploiement automatisé.

En récapitulue que, l'objectif principal de DevOps est de favoriser la communication et la collaboration entre les équipes de développement et les équipes d'exploitation afin de mettre en place un processus complet et automatisé pour le déploiement de produits performants.

Ainsi, les avantages de ce concept seront résumés par :

- Accélération du temps de démarrage.
- Réduction de risque.
- Accélération de la réponse aux incidents.
- Amélioration de la satisfaction client.

Pratique DevOps

Les pratiques DevOps, représentées par la figure 2.1 seront englobées par deux grandes phases CI (intégration continue) / CD (déploiement continu). Chaque pratique sera responsable de l'automatisation continue des processus de déploiement d'applications avec des améliorations continues.

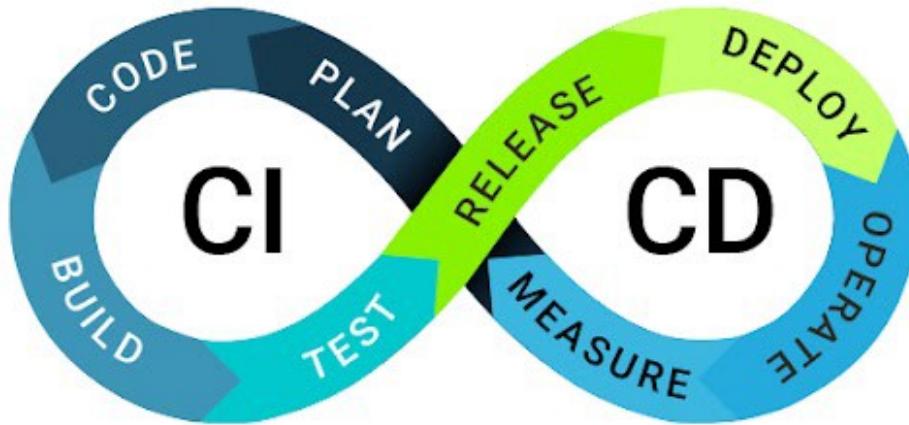


FIGURE 2.1 – Les pratiques DevOps

Développement continu : Cette pratique englobe les phases de planification et de codage du cycle de vie DevOps qui peuvent inclure des mécanismes de contrôle de version (GIT, SVN).

Intégration continue (Build) : Cette pratique rassemble des outils de gestion de configuration et de développement pour suivre la publication de différentes parties de code.

Tests continus : Cette pratique prévoit des tests automatisés, planifiés et continus lors de l'écriture ou de la mise à jour du code d'application qui accélère la livraison du code à la production.

Livraison continue : Cette pratique automatise la publication des modifications de code après la phase de test, dans un environnement intermédiaire.

Déploiement continu : Comme la livraison continue, cette pratique automatise la publication d'un nouveau code ou un code modifié dans l'environnement de production.

Surveillance continue : Cette pratique permet une surveillance continue du code exécuté et de l'infrastructure sous-jacente. Les développeurs reçoivent des commentaires sur les bugs ou les problèmes.

Infrastructure-as-code : Cette pratique peut être suivie en plusieurs phases DevOps pour automatiser le provisionnement de l'infrastructure requise pour une version logicielle.

Afin de réussir ce concept et facilite l'automatisation de déploiement des nouveaux produits livrables, nous avons besoin d'un processus d'intégration continue / déploiement continu (CI/CD).

2.2.2 Intégration Continue / Déploiement Continue

L'approche CI/CD, illustrée dans la figure 2.2, garantit l'automatisation de développement des applications, en instaurant des éléments de surveillance pour s'assurer que l'application fonctionne bien.

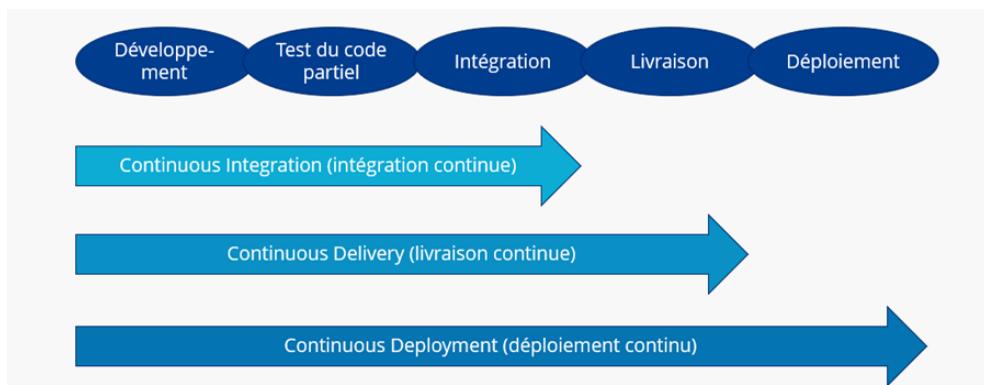


FIGURE 2.2 – L'approche CI/CD

Intégration Continue

Le développement de logiciels implique que plusieurs développeurs travaillent simultanément sur différentes tâches de la même application. Ils fusionnent ensuite leur travail pour créer un livrable au client, ce qui prend beaucoup de temps et peut provoquer de multiples conflits avec d'autres codes à chaque modification.

Par conséquent, nous utilisons le concept d'intégration continue (CI), une pratique de développement qui oblige les développeurs à intégrer régulièrement leurs modifications de code dans un référentiel partagé à l'aide des branches, puis à vérifier chaque commit (ou branche) via une compilation automatique.

L'objectif ultime de l'intégration continue est de fusionner le code de différents développeurs, d'effectuer les tests unitaires nécessaires et de créer le code pour fournir un meilleur produit plus rapidement, permettant aux équipes de trouver et de résoudre rapidement les problèmes afin de réduire le temps nécessaire pour valider et publier de nouvelles mises à jour.[?]

Livraison continue

Après avoir automatisé les constructions (Builds), les tests unitaires et d'intégration dans le cadre de l'intégration continue, la distribution continue automatise la publication du code validé dans un référentiel. Le processus de distribution continue aide à résoudre les problèmes de visibilité et de communication entre les équipes de développement et d'exploitation. Par conséquent, son objectif est de déployer le nouveau code en production aussi facilement que possible.[?]

Déploiement Continue

Le déploiement continu est le processus de lancement automatique d'une application dans un environnement de production pouvant être utilisé par les clients.

Ce processus soulage les équipes d'exploitation qui sont surchargées de tâches manuelles qui ralentissent la livraison des applications. Il est basé sur la livraison continue et l'automatisation des prochaines étapes d'automatisation. À ce stade, il n'y a aucun moyen d'empêcher le déploiement de nouvelles modifications autres que l'échec du test.[?]

Les principaux outils qui seront illustrés dans cette phase sont : - le docker pour containeriser notre application et le kubernetes pour orchestrer les conteneurs déployer.

Ces trois phases seront organisées dans un script appelé « Pipeline » et qui sera détailler dans la partie 2.2.3.

2.2.3 Pipeline CI/CD

Un pipeline CI/CD est une suite d'étapes à réaliser afin de distribuer une nouvelle version d'un logiciel basée sur le monitoring et l'automatisation pour améliorer le processus de développement, notamment dans les phases d'intégration et de test, ainsi que pendant la distribution et le déploiement à l'aide de l'approche DevOps,

Les étapes d'un pipeline CI/CD :

- **Source** : Stocker le code dans un référentiel décentralisé à l'aide d'un système de contrôle de version, tels que Git.
- **Test** : Découvrir les problèmes au cours des étapes de développement.
- **Construction** : Combiner le code source avec des dépendances pour obtenir une instance du logiciel qui sera finalement prête pour les utilisateurs finaux.
- **Déploiement** : Déployer du code en production.

2.3 Architecture générale proposée

Passons maintenant à la deuxième partie de ce chapitre, où nous présenterons l'architecture générale de notre solution proposée - illustrée à la figure 2.3 - qui comprend la mise en place d'un pipeline CI/CD pour déployer notre application.

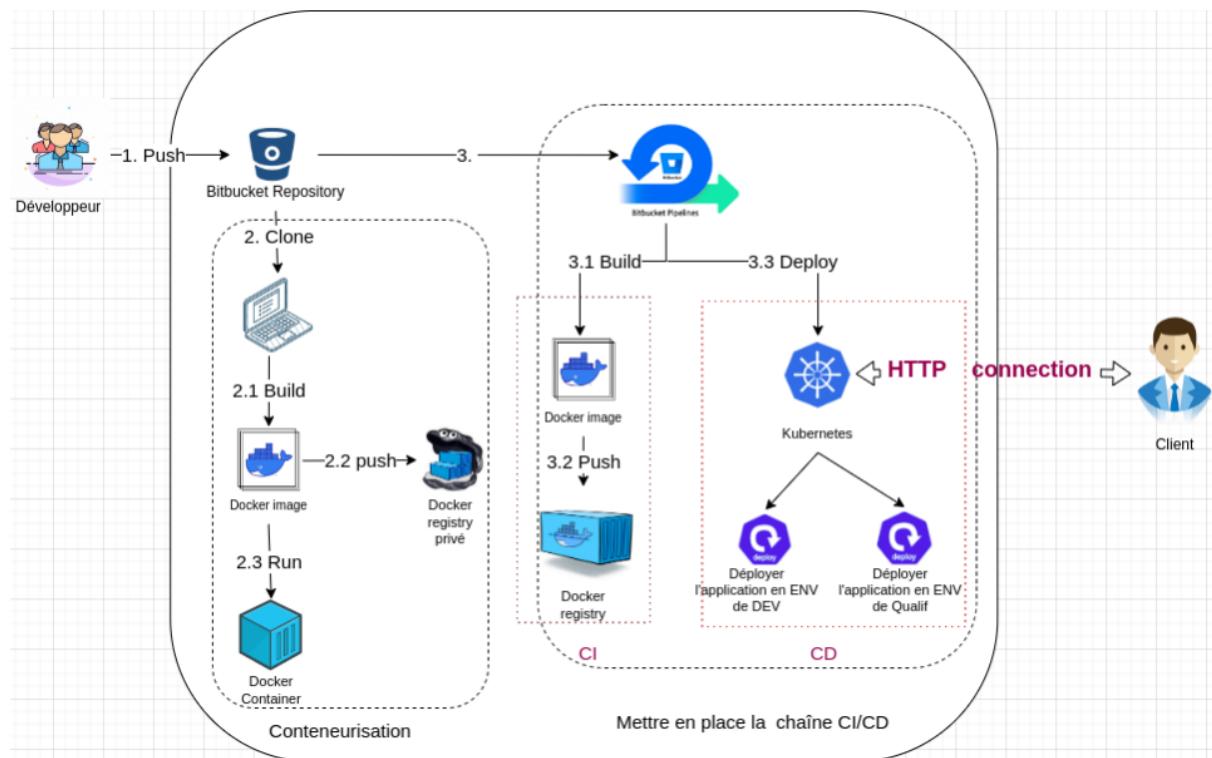


FIGURE 2.3 – Architecture Générale proposée

Au début, l'équipe DEV renvoie (push) le code dans le répertoire d'outil de versioning adopté comme le Bitbucket dans notre cas -1-. À cette phase, le travail de l'équipe de développement se termine et l'équipe opérationnelle se prépare à boucler le cycle restant phases : **l'Intégration continue et le Déploiement continu**.

Passons maintenant à la première exécution du travail de l'équipe OPS : **la conteneurisation** -2.- qui sera présentée en détail au chapitre 3. Dans cette partie sera exécuté par plusieurs étapes :

- ⇒ À partir de l'étape - 2. -, nous allons cloner le code source situé dans le répertoire 'bitbucket' en local.
- ⇒ À l'étape suivante - 2.1 -, nous construirons (Build) l'image docker de notre application, qui sera stockée (push) dans un registre docker privé - 2.2 -.
- ⇒ Dans la dernière étape - 2.3 -, nous déployerons l'application localement avec les conteneurs docker.

La deuxième partie qui sera présentée au chapitre 4 :**Déploiement des micro-services sur GKE**. Dont l'objectif de cette division est d'automatiser le déploiement de notre application à l'aide du pipeline CI/CD.

- ⇒ Au début, nous exécuterons la phase d'intégration continue(CI), qui comprend la création de l'image docker (build) - 3.1 - ainsi que son insertion dans un répertoire docker (push) - 3.2 -.
- ⇒ Ensuite nous passerons à la phase de déploiement Continu(CD) avec kubernetes, notre solution sera déployée sur deux environnements :

L'environnement de « **DEV** » qui est alloué aux équipes de développements pour tester le fonctionnement du code.

L'environnement « **Qualif** » qui est organisé pour que les équipes de tests réalisent les tests nécessaires avant de déployer le produit sur l'environnement de production, qui est l'environnement final accessible aux clients finaux.

2.3.1 Les outils à utiliser

Généralement, les outils de gestion des versions sont utilisés dans les processus de développement de logiciels et la gestion de contenu sur le lieu de travail. D'où, les objectifs d'un tel outil sont, d'une part, de coordonner l'accès commun de plusieurs utilisateurs aux fichiers et, d'autre part, de permettre le développement simultané de plusieurs branches.

Les outils de contrôle de version les plus connus sont Apache Subversion (SVN) et Git.[?]

SVN

SVN s'appuie sur un système de gestion des versions centralisé. Cela signifie qu'un seul répertoire général existe et que tous les utilisateurs y ont accès. Étant donné que les modifications ne peuvent pas être fusionnées.

Ce système empêche deux utilisateurs de modifier le même fichier simultanément, il l'attribue au premier internaute qui l'ouvre et il reste protégé des autres utilisateurs tant qu'il n'a pas été fermé.[?]



FIGURE 2.4 – SVN

Git

Git est un système de contrôle de version décentralisé créé en 2005 par Linus Torvalds, distribué pour suivre les modifications du code source pendant le développement de logiciels. Il est conçu pour coordonner le travail entre les programmeurs. En outre, il peut être utilisé pour suivre les modifications des fichiers de n'importe quel projet. Ses objectifs incluent la vitesse, l'intégrité des données et la prise en charge des flux de travail distribués et non linéaires.[?]



FIGURE 2.5 – Git

Étude Comparative

Dans le tableau 2.1 nous présenterons une comparaison entre Git et SVN par plusieurs critères pour bien justifier notre choix.

SVN	Git
	Versionning
Centralisé	Décentralisé
	Dépôt
Utilise un dépôt central dans lequel les copies de travail sont créées	Des copies de dépôt, présentes localement, dans lesquelles il est possible de travailler.
	Droit d'accès
Basé sur le chemin	Pour le répertoire complet
	Suivi des modifications
Enregistre des données	Enregistre des données.
	Connectivité au réseau
Pour tous les accès	Nécessaire seulement pour réaliser une synchronisation.
	Journal de modifications des données
Complet seulement dans le dépôt. Les copies de travail ne contiennent que la version la plus récente.	Le dépôt et les copies de travail contiennent l'historique complet.

TABLE 2.1 – Comparaison entre SVN et Git

Par conséquent, nous choisissons le Git pour commiter les nouveaux changements durant nos exécutions, créer des branches, faire des merges et comparer les anciennes versions.

2.3.2 Virtualisation

La virtualisation est une technologie qui vous permet de créer plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique. Elle se présente lors de la production des applications sous la forme de deux modèles : la virtualisation avec des machines virtuelles et la conteneurisation.

La figure 2.6 présente la différence entre la conteneurisation et la virtualisation avec des machines virtuelles au niveau des architectures :

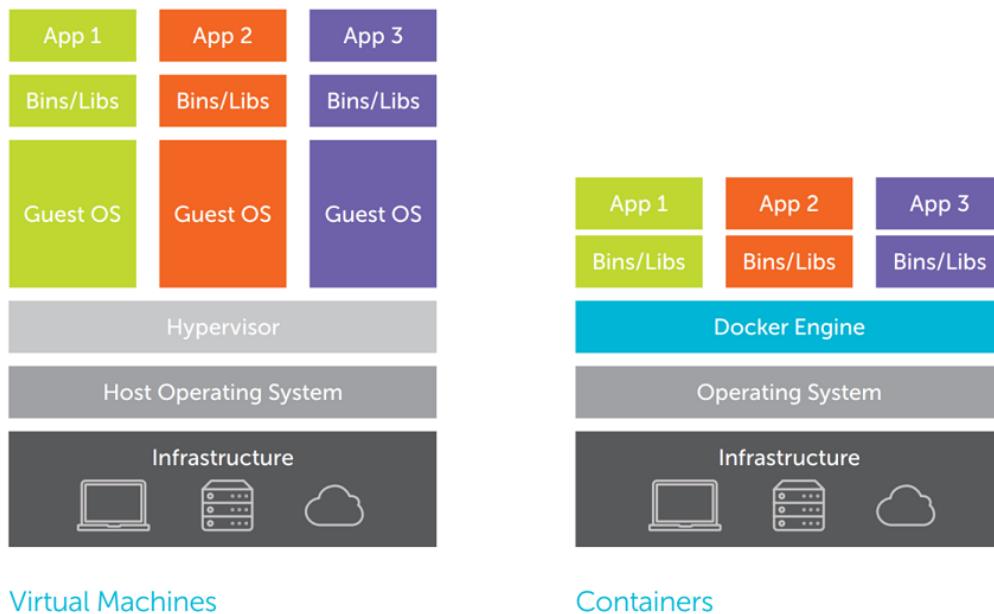


FIGURE 2.6 – Architectures des machines virtuelles

Les machines virtuelles

La virtualisation est une technologie qui vous permet de créer plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique. Avec l'hyperviseur (logiciel de virtualisation) qu'est directement relié au matériel et vous permet de fragmenter ce système unique en plusieurs environnements sécurisés distincts. C'est ce que l'on appelle les machines virtuelles.

Ces derniers permettent d'exécuter simultanément plusieurs systèmes d'exploitation sur un seul ordinateur, et chacun de ces systèmes s'exécute sur le matériel hôte comme le ferait n'importe quel autre système d'exploitation ou application [?]

Conteneurisation

La conteneurisation est devenue le dernier mot à la mode dans le cloud computing, et beaucoup pensent qu'elle peut aider à moderniser les systèmes existants en créant de nouvelles applications évolutives conçues dans le cloud.

Il s'agit d'une forme de virtualisation du système d'exploitation dans laquelle, elle permet l'exécution des applications dans des espaces utilisateur isolés appelés conteneurs qui utilisent le même système d'exploitation partagé (Kernel) en permettant aux développeurs de logiciels de créer et de déployer des applications plus rapidement et de manière plus sécurisée. [?]

Étude Comparative

Dans le tableau 2.2 nous illustrons la différence entre la virtualisation avec les Vms et la conteneurisation.

	Virtualisation	Conteneurisation
Portabilité et Légèreté	- Chaque application est codée sur un environnement informatique spécifique en occupant beaucoup d'espace disque (chaque VM à son propre OS complet)	- Création d'un logiciel exécutable qui est isolé par rapport au système d'exploitation hôte et en conséquent son très faible poids
Performance du système	- Lors de l'exécution d'applications intégrées dans des machines virtuelles, l'utilisation de la mémoire peut être plus élevée que nécessaire donc elles augmentent la pression sur la mémoire de l'hôte	- Les conteneurs partagent un environnement de système d'exploitation (noyau), donc elles utilisent moins de ressources et réduisent la pression sur la mémoire de l'hôte.
Maintenance et mises à jour	- Chaque système d'exploitation invité doit être corrigé séparément.	- Seul le système d'exploitation de l'hôte du conteneur doit être mis à jour

TABLE 2.2 – Comparaison entre Virtualisation et Conteneurisation

En effet, nous constatons que la conteneurisation offre plusieurs avantages par rapport à la virtualisation traditionnelle (avec les machines virtuelles), d'où nous choisissons le concept des conteneurs pour réaliser notre projet.

Le cloud computing

La bonne implémentation de la méthode DevOps vise à réduire la latence et l'inefficacité du code lors des phases de développement en utilisant l'automatisation depuis la phase de création (Intégration Continue) jusqu'à l'exploitation (Déploiement Continu) en passant par la phase de maintenance (Tests Continus). Ce concept a besoin des environnements pour exécuter ces pratiques qui sont fournies par le cloud. En fait, DevOps a besoin de tout ce que le cloud a à offrir.[?]

Les modèles de services de Cloud

Les services cloud prennent la forme d'infrastructures (IaaS), de plateformes (PaaS) ou de logiciels (SaaS), hébergés par des fournisseurs tiers et accessibles aux utilisateurs via Internet. La figure 2.7 illustre les modèles cloud.

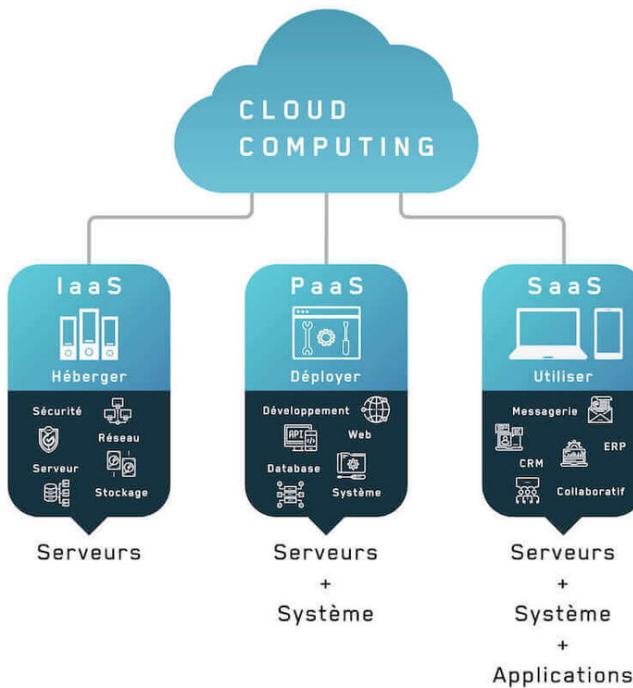


FIGURE 2.7 – Modèles de services de Cloud

- **Software-as-a-Service(SaaS) :** Les applications SaaS sont hébergées sur des serveurs cloud et les utilisateurs y accèdent via Internet pour les installer sur leurs appareils. Parmi les exemples d'applications SaaS, on citera Salesforce, MailChimp et Slack.
 - **Platform-as-a-Service(PaaS) :** Permet aux entreprises de ne payer que les éléments dont elles ont besoin pour créer leurs propres applications. De plus, les fournisseurs PaaS offrent tout le nécessaire pour créer une application, y compris les outils de développement, l'infrastructure et les systèmes d'exploitation. Parmi les exemples de services PaaS, on citera notamment Heroku et Microsoft Azure.
 - **Infrastructure-as-a-Service(IaaS) :** L'entreprise loue les serveurs et l'espace de stockage dont elle a besoin auprès d'un fournisseur de cloud pour développer ses propres applications. DigitalOcean, Google Compute Engine et OpenStack sont des exemples de fournisseurs de services IaaS.
- <https://www.cloudflare.com/fr-fr/learning/cloud/what-is-the-cloud/>

Les types de déploiements Cloud

Il existe trois types de déploiements cloud qu'une organisation peut utiliser, illustrés à la Figure 2.8

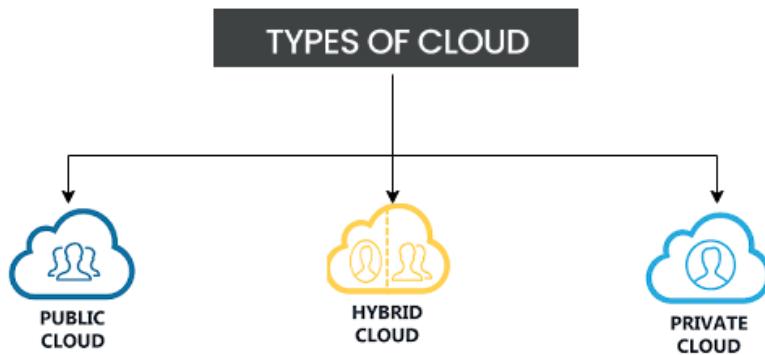


FIGURE 2.8 – Les types de cloud

- **Cloud Privé :** Le cloud privé fait référence à un serveur, un datacenters ou un réseau distribué, entièrement dédié à une organisation protégée derrière un pare-feu, qui est un système de sécurité qui suit et contrôle le trafic réseau.
- **Public Cloud :**
Public Cloud fait référence à un service géré par un fournisseur externe et pouvant inclure des serveurs localisés dans un ou plusieurs datacenters. De plus, les clouds publics sont partagés par de nombreuses organisations.
- **Cloud hybride :**
Les déploiements de cloud hybride combinent des clouds publics et privés. Ainsi, une organisation peut utiliser son cloud privé pour certains services et son cloud public pour d'autres, à moins qu'elle ne préfère conserver son cloud public en cas de panne de son cloud privé.

Les fournisseurs cloud

Les fournisseurs de cloud proposent de nombreux services pour réduire les coûts de développement et de déploiement des applications. Il existe plusieurs fournisseurs de cloud que nous énumérerons dans ce rapport :

- **Amazon Web Services**

En 2005, Amazon s'est lancé dans une nouvelle activité consistant à fournir des plateformes de cloud computing appelées Amazon Web services. Depuis lors, l'entreprise fournit à ses clients des serveurs, en fonction de leur demande et de leur évolution. Par exemple, si une entreprise est témoin d'un bond énorme sur un serveur, AWS lui fournit d'autres serveurs afin qu'elle puisse gérer le trafic en ligne de manière transparente. Une fois que cela est fait, AWS libère ses serveurs pour d'autres entreprises qui en ont besoin. De cette façon, AWS aide les entreprises à gérer leurs données.[?]



FIGURE 2.9 – Amazon Web Services (AWS)

→ Google Cloud Platform

Google Cloud Platform (GCP) est une suite de services d'informatique en cloud proposée par Google LLC. GCP a vu le jour en 2008 et offre divers services allant de l'informatique au stockage, en passant par la mise en réseau et l'IA en cloud, entre autres. Une chose qui rend GCP unique est qu'il a développé TensorFlow, un logiciel gratuit et open-source utilisé par la plupart des programmes de machine learning.[?]



Google Cloud

FIGURE 2.10 – Google Cloud Plateform

Nous présentons la comparaison entre « AWS » et « GCP » dans le tableau suivant, 2.3 :

	AWS	GCP
Part de marché	AWS est en tête avec une marge énorme, suivi de Microsoft Azure et de Google Cloud	Au 4ème trimestre 2021, la part de marché de Google Cloud est de 9 % au niveau mondial est l'une des plus grandes entreprises de cloud au monde.
Géographie et Zones de disponibilité	Le réseau d'AWS s'étend sur 245 pays et territoires et est disponible dans 77 zones (au Japon, en Inde, en Espagne et en Suisse)	Google Cloud Network est disponible dans 24 régions et 73 zones, touchant la base de plus de 200 pays (en Asie occidentale et en Europe)
Capacités Open Source	Amazon n'a pas de programmes adaptés aux logiciels libres.	GCP a une longueur d'avance en ce qui concerne la contribution de l'open source.
Services de stockage	Service de stockage similaire en proposant un stockage des réseaux en cloud	
Services de mise en réseau	AWS permet aux utilisateurs d'établir une connexion directe avec d'autres partenaires AWS grâce à Direct Connect	GCP offre aux utilisateurs la possibilité d'établir une interconnexion avec d'autres partenaires GCP à l'aide du service Interconnect.
Prix	le modèle de tarification d'Amazon Web Services est coûteux car il facture les clients sur une base horaire,	Google Cloud Platform est l'une des plateformes les plus rentables offrant un large éventail de fonctions sans frais supplémentaires.

TABLE 2.3 – Comparaison entre AWS et GCP

Suite à cette comparaison, nous choisissons de travailler avec la plateforme Google Cloud Platform.

Les avantages du DevOps basé sur le cloud

- **Options d'automatisation centrées sur le cloud :** L'automatisation est un outil essentiel pour rendre DevOps efficace. Il existe plusieurs plates-formes cloud qui fournissent des outils d'automatisation avancés via des processus DevOps tels que CircleCI, Jenkins, GitLab et Travis CI qui sont responsables du développement continu et de l'intégration continue.
- **Plateforme centralisée :**
Le cloud fournit une plate-forme centralisée où les entreprises peuvent tout gérer, depuis les tests, le déploiement, la surveillance et le déploiement de leurs charges de travail de production. Cela facilite le suivi de tout dans le même espace.
- **Infrastructure évolutive :**
Le cloud est le meilleur moyen de vous assurer que vous pouvez mettre à niveau ou rétrograder n'importe quelle infrastructure selon vos besoins, sans débourser beaucoup d'argent et de temps sur les systèmes. Cette évolutivité signifie que DevOps

est une solution très efficace pour déployer de nouvelles fonctionnalités.

→ **Disponibilité et stabilité :**

Parce que les fournisseurs de cloud se concentrent sur la disponibilité et la stabilité, ils peuvent gérer tous les éléments de gestion et de maintenance de la plateforme. Plutôt que de se soucier de ces problèmes, les entreprises informatiques peuvent se concentrer sur le développement de produits pour améliorer les performances des produits, améliorer l'expérience utilisateur et accélérer la mise sur le marché.[?]

2.3.3 Les orchestrateurs de conteneurs

L'orchestration des conteneurs responsable d'automatiser le déploiement, le ména-gement, et la configuration et la mise en réseau des conteneurs. Parmi ces outils nous citerons : **Docker** et **Kubernetes**.

Docker

Docker est un outil conçu pour faciliter la création, le déploiement et l'exécution d'applications à l'aide de conteneurs.

Ce logiciel a été développé sur la base de la technologie Linux Container (LXC), qui a ensuite été remplacée par le libcontainer de Docker. Par conséquent, des nouveaux composants logiciels ont été ajoutés pour que Docker continuait de croître et devenait l'un des normes de virtualisation basée sur des conteneurs.

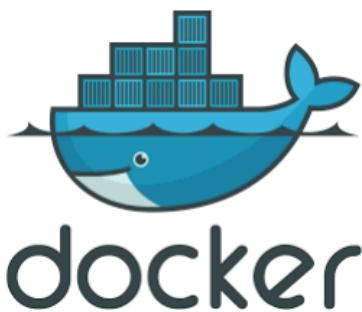


FIGURE 2.11 – Docker

D'une certaine manière, Docker est un peu comme une machine virtuelle. Mais contrairement à une machine virtuelle, plutôt que de créer un système d'exploitation virtuel complet, Docker permet aux applications d'utiliser le même noyau Linux que le système sur lequel elles s'exécutent.

Par conséquent, il améliore considérablement les performances et réduit la taille de l'application.

[?]

Les composants docker

La plateforme Docker repose sur plusieurs composants, qui sont représentés par la figure 2.12 :

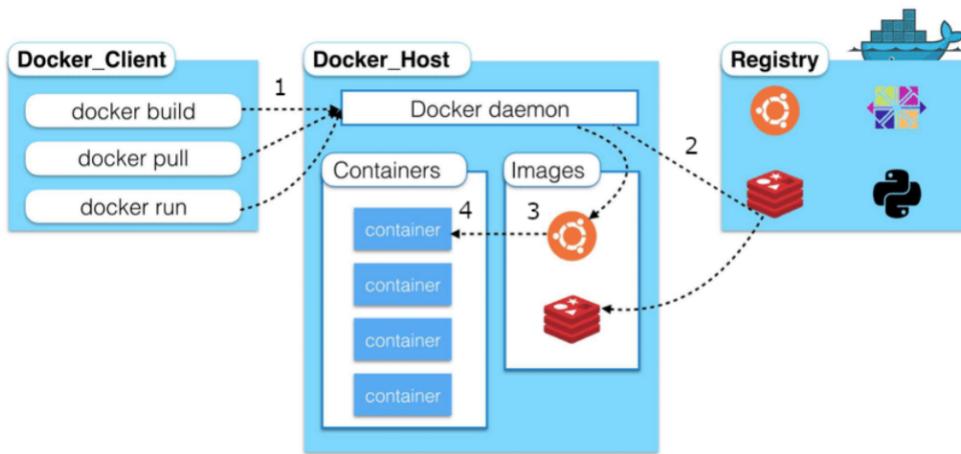


FIGURE 2.12 – Les composants de Docker

→ Docker Client

Permet l'interaction entre les utilisateurs et Docker. Lorsqu'une commande docker s'exécute, le client les envoie au démon docker, qui les exécute. Le client Docker peut communiquer avec plusieurs démons.

→ Docker Engine

C'est la partie centrale de tout le système Docker. Consiste à une application qui suit une architecture client-serveur, et qui s'installe sur la machine hôte. Il existe trois composants dans le Docker Engine :

- ★ Serveur : Démon docker appelé dockerd. Créer et gérer des images Docker, Conteneurs, réseaux...
- ★ API REST Utiliser pour dire au démon docker quoi faire.
- ★ Interface de ligne de commande (CLI) : C'est le client qui utilise les commandes docker.

→ Docker registry

C'est l'emplacement où les images Docker sont stockées. Il peut s'agir d'un registre docker public ou d'un registre docker privé. Docker Hub est l'emplacement par défaut des images docker, le registre public de ses magasins.

Kubernetes

Kubernetes est une plateforme open source d'orchestration de conteneurs conçue à l'origine par Google en 2015. Ce projet permet d'automatiser le déploiement et la gestion des applications multi-container.



FIGURE 2.13 – Kubernetes

Kubernetes a un certain nombre de fonctionnalités. Il peut être considéré comme :

- une plate-forme de conteneurs.
- une plate-forme de microservices.
- une plate-forme cloud portable et beaucoup plus.

Kubernetes fournit un environnement de gestion focalisé sur le conteneur (container-centric). Il orchestre les ressources machines (computing), la mise en réseau et l'infrastructure de stockage sur les workloads(charges de travail) des utilisateurs.

[?]

Par conséquent, cet outil à plusieurs avantages, comme :

- ✓ Sa rapidité de déploiement.
- ✓ L'organisation facile du service à l'aide de ces objets.
- ✓ Sa variété d'options de stockage, y compris les SAN locaux et les clouds publics.

Les objets de kubernetes

L'objectif principal de Kubernetes est d'éviter la complexité de gestion d'un groupe de conteneurs, et pour cela, différents objets existent pour nous faciliter le déploiement de notre application, tels que :

→ **Pod :**

Les pods sont les plus petites unités informatiques déployables que vous pouvez créer et gérer dans Kubernetes. C'est un groupe d'un ou plusieurs conteneurs, avec un stockage partagé et des ressources réseaux, ainsi qu'une spécification sur la manière

d'exécuter les conteneurs.

[?]

→ **Service :**

Une attitude abstraite consistant à présenter une application s'exécutant sur un ensemble de pods. Permet d'allouer en permanence des adresses IP et DNS pour les pods et de les mettre à jour dynamiquement.

→ **Namespace :**

C'est un regroupement d'un ensemble des composants.

→ **Contrôleurs :**

Définit l'état souhaité de cluster et garantit l'état souhaité d'un pod :

★ Deployment : C'est une description permettant la création d'une infrastructure complète (avec Pod, Service). Le fichier Deployment est le plus gros fichier que vous pourriez avoir si vous faites un déploiement sur K8S.

★ REPLICASET : un replicaset permet de mettre à l'échelle un déploiement.

→ **Volumes :**

C'est considéré comme un répertoire accessible aux conteneurs d'un pod. Il existe différents types de volumes :

★ Un PersistentVolume (PV) : Un élément de stockage au niveau de cluster, provisionné par un administrateur ou dynamiquement.

★ Un PersistentVolumeClaim (PVC) : Une demande de stockage par un utilisateur qui est similaire à un Pod. Les pods consomment des ressources de noeud et les PVC consomment des ressources PV.

Les composants de Kubernetes

→ **Cluster kubernetes**

C'est un ensemble de noeuds (les machines) qui permettent d'exécuter des applications conteneurisées. Le cluster Kubernetes englobe deux principaux composants : le plan de contrôle et les noeuds (les machines de calcul).

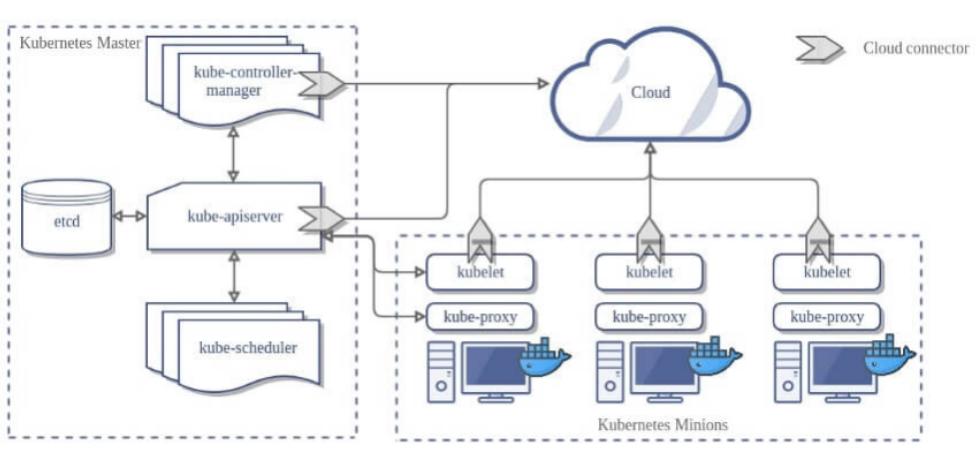


FIGURE 2.14 – Cluster kubernetes

→ **Kubectl :**

Permet aux utilisateurs d'interagir avec Kubernetes à l'aide d'un outil CLI appelé Kubectl ou de toute interface utilisateur.

→ **Nœud Master (Control-plan) :**

Le Nœud Master est responsable du maintien de l'état souhaité pour chaque cluster. Il est composé de 4 éléments :

- ★ Le serveur d'API (API Server) : c'est tout ce qui va être exposé pour que vous puissiez faire vos déploiements.
- ★ Etcd : représente la base de données de Kubernetes en stocke les états du système.
- ★ Scheduler : Il est chargé de répartir la charge de travail sur les nœuds du cluster en fonction des ressources nécessaires et de celles disponibles.
- ★ Le contrôleur (Controller Manager) : C'est le superviseur Kubernetes, qui vérifie à chaque fois que l'utilisateur saisit une commande, si elle a été exécutée correctement et comment elle a été exécutée.

→ **Worker Nodes**

Ce sont des composants qui exécutent les applications et les tâches assignées par le master nœud.

- ★ Kubelet : c'est un service responsable au démarrage des pods sur les nœuds.

- ★ Kube-Proxy : mise en réseaux des pods.
- ★ Container runtime : Environnement d'exécution des conteneurs placés dans les pods.

2.4 Conclusion

Ce chapitre est réservé dans une première partie à l'étude préliminaire de notre projet où nous montrons les principales clés de notre solution : DevOps et l'approche d'intégration continue / déploiement continu. Et dans la deuxième partie, nous avons présenté l'architecture globale de notre solution, ainsi que les éléments clés adoptés, qui seront détaillés dans les chapitres suivants.

Dans le chapitre suivant – chapitre 3 – nous détaillerons le concept de conteneurisation en présentant la partie pratique à réaliser lors de notre stage.

Chapitre 3

Conteneurisation

3.1 Introduction

Ce chapitre sera consacré à détailler le premier volet de notre projet : la conteneurisation.

Commençant par une présentation détaillée de l'architecture proposée, nous définirons ensuite l'environnement de travail qui fera l'objet d'une étude théorique de ce chapitre, dont le but est de faciliter la compréhension de la partie pratique, où nous présenterons les traces d'exécution.

En effet, l'objectif de ce chapitre est le déploiement de notre application avec docker.

3.2 Architecture de conteneurisation proposée

Notre application est composée de trois micro-services : le micro-service front développé en ReactJS, le micro-service API développé en NodeJS et le micro-service BackEnd développé en Strapi lié à une base de données MongoDB.

Au début, nous créerons un fichier d'instructions ‘Dockerfile’ qui est responsable à la construction(build) d'une image docker à chaque micro-service, qui sera hébergée dans un répertoire docker privé « Harbor ».

Juste après, nous allons déployer les images docker obtenues dans des conteneurs, qui seront orchestrées par l'outil d'orchestration docker-compose.

Finalement, nous exposerons nos micro-services par un nom de serveur (server_name) bien déterminé à l'aide du serveur web ‘Nginx’.

La figure 3.1 montre notre architecture proposée :

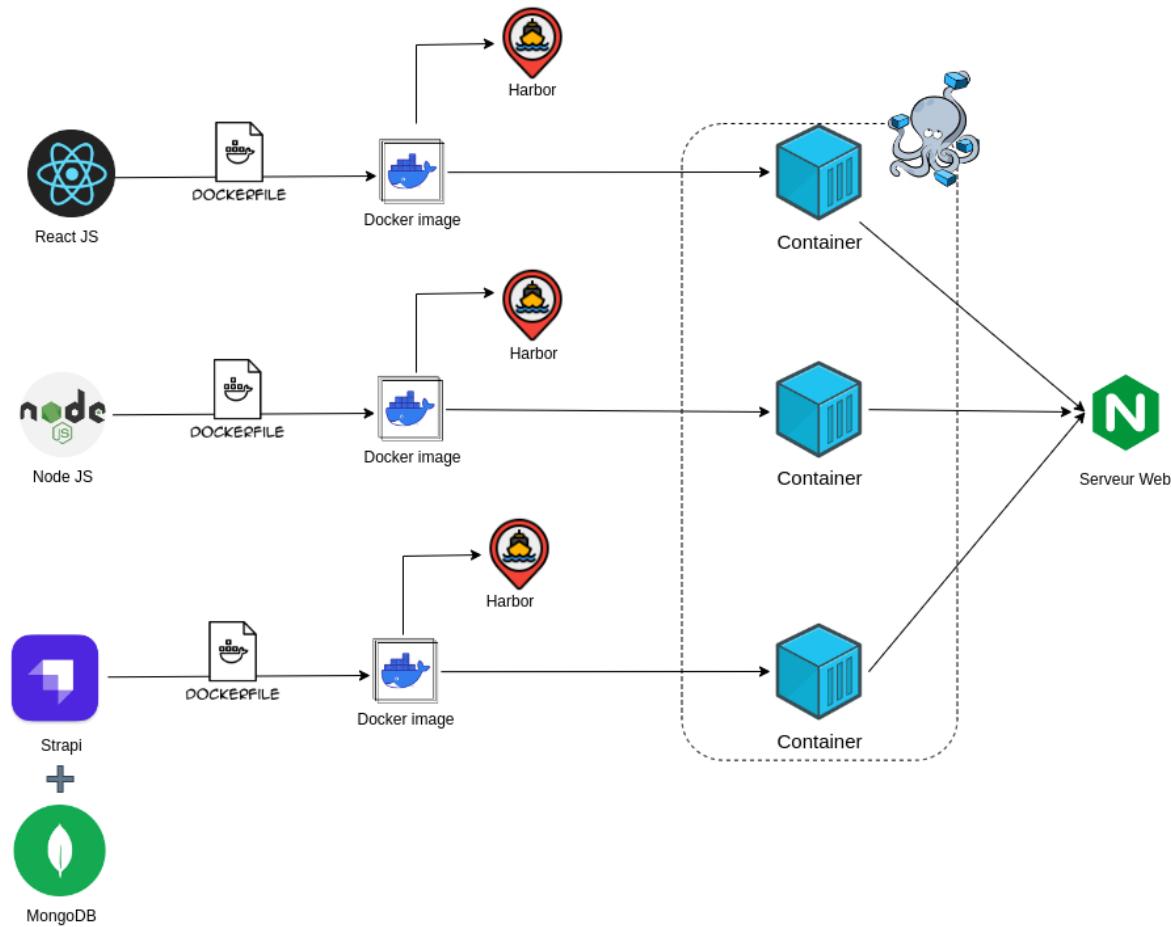


FIGURE 3.1 – Architecture de conteneurisation proposée

3.3 Environnement du travail

Durant cette partie, nous définirons les différents outils et technologies adoptées, pour atteindre l'objectif de ce chapitre, commençant par les objets de docker (Dockerfile, Docker image, les conteneurs), Docker-compose, Harbor en tant que registre docker, et finalement le serveur web utilisé.

3.3.1 Les objets de docker

Docker est connu comme une technologie de conteneurisation par sa gestion des dépendances au sein des projets lors de sa construction (Build) et de son déploiement (Run), en utilisant ces principaux objets : docker image, Dockerfile et containers, ainsi que des volumes.

Ces différents objets seront présentés dans la figure 3.2 :

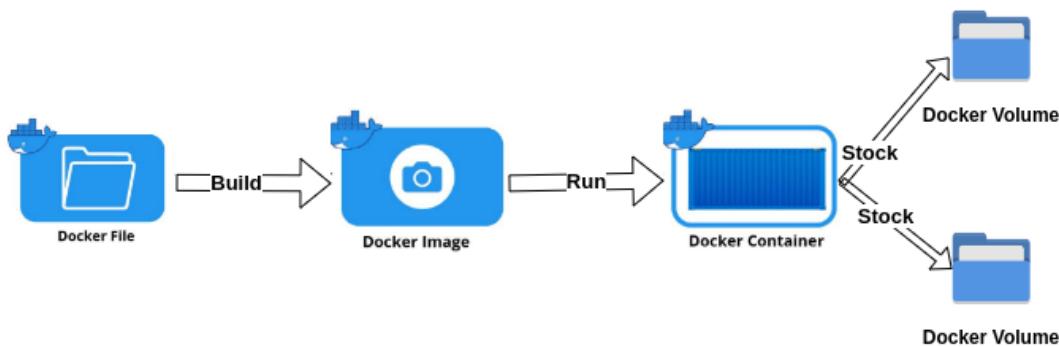


FIGURE 3.2 – Les objets Docker

⇒ **Dockerfile :**

C'est un fichier texte qui décrit la structure d'une image Docker, similaire à un script de traitement par lots qui contient des commandes décrivent une image. Lors de l'exécution de ce fichier, les commandes sont traitées l'une après l'autre.

Nous pouvons synthétiser Dockerfile comme une sorte de recette utilisée comme base pour créer une image.

⇒ **Docker image :**

Une image Docker est un paquet exécutable en lecture seule utilisée à la construction d'un ou plusieurs conteneurs identiques.

⇒ **Les conteneurs :**

Les conteneurs permettent à un développeur de regrouper une application avec toutes les parties dont elle a besoin, telles que des bibliothèques et d'autres dépendances, et de l'exporter dans un seul package.

De plus, il s'agit d'un environnement léger, autonome et isolé qui garantit au développeur que l'application s'exécutera sur n'importe quelle autre machine Linux, quels que soient ses paramètres personnalisés qui pourraient différer de la machine utilisée pour l'écrire et l'exécuter.

L'objectif principal de l'utilisation d'un conteneur est de tester des applications en développement, ainsi que des logiciels.

⇒ **Volume Docker :**

Ce volume est placé dans l'ordinateur hôte, en dehors du véritable conteneur. Le volume joue le même rôle qu'un dossier où nous pouvons stocker ou extraire des données.

3.3.2 Docker-Compose

Docker Compose est un outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services inter-connectés,

qui correspondent aux options disponibles lors d'un docker run : l'image à utiliser, les volumes à monter, les ports à ouvrir, etc.

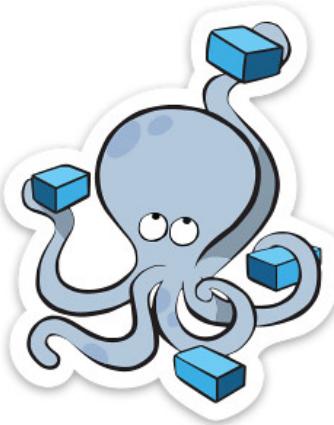


FIGURE 3.3 – Docker-Compose

3.3.3 Registre Docker

Généralement, les registres sont semblables à des bibliothèques où les images sont stockées et peuvent être extraites pour générer des conteneurs afin d'exécuter les services ou les applications web.

Il existe deux types de registres : les registres Docker privés et les registre public comme ‘Docker hub’.

[?]

Dans notre situation nous concentrerons sur le registre docker privé ‘Harbor’. Ce dernier est une registre docker privée Open Source avec un licence Apache 2.0 créé et maintenu par VMWare, soutenu aussi par la Cloud Native Computing Foundation.

Ce registre permet d'avoir un endroit en interne pour stocker et scanner nos propres images ou directement les images issues de Docker Hub par exemple, tout en proposant une interface graphique lisible et claire.

[?]



FIGURE 3.4 – Harbor

3.3.4 Les serveurs web

Un serveur Web est un modèle de programme client/serveur qui utilise le protocole HTTP pour fournir les fichiers qui composent les pages Web demandées par les utilisateurs.

Tous les ordinateurs qui hébergent des sites Web doivent avoir des programmes de serveur Web. Les principaux serveurs Web les plus connus sont Apache, et Nginx de NGINX qui sera utilisé dans notre projet.

Nginx a été créé à l'origine par Igor Sysoev, avec sa première sortie publique en octobre 2004 dont l'objectif est de créer un serveur de haute performance qui peut gérer en même temps plusieurs clients web.

Le but du serveur Nginx est de permettre une faible utilisation de la mémoire et une grande simultanéité.



FIGURE 3.5 – Nginx

Ce programme a des différentes fonctionnalités :

- **Reverse Proxying ou proxy inverse** : ici, Nginx est utilisé comme un proxy inverse dans le but d'accélérer les chargements web ou l'utilisation d'un proxy email.
- **Application Acceleration** : cette fonction permet de charger rapidement les contenus.
- **Load balancing ou équilibrage de charge** : celle-ci diminue les charges sur le serveur principal en répartissant les demandes.
- **Chiffrement TLS** : cette fonctionnalité sécurise les échanges de données.
- **Gestion de la bande passante** : cette fonction associe la bande passante optimale pour chaque service proposé.
- **Videostreaming ou vidéo en streaming** : elle donne une meilleure performance pour le streaming de fichiers M et FLV.

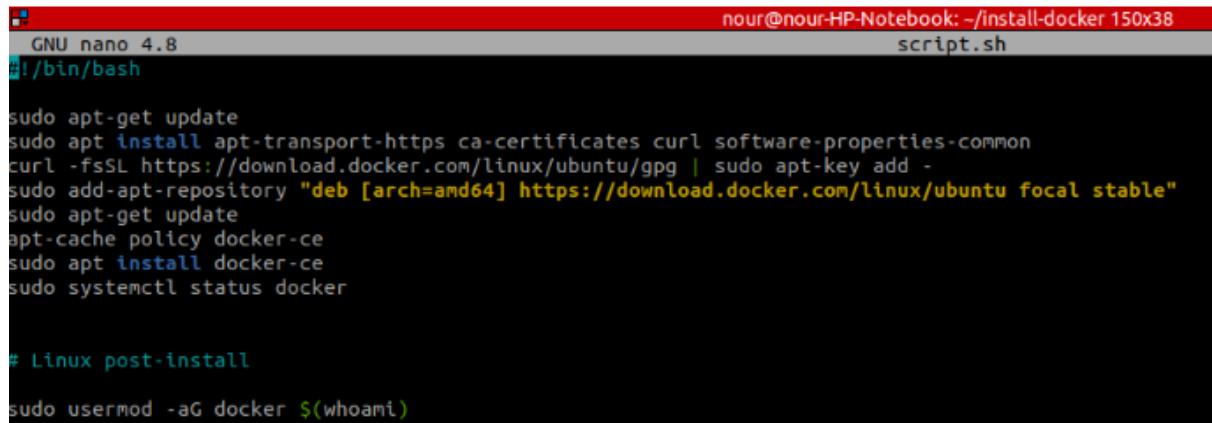
3.4 Conteneurisation du projet

Après avoir présenté les outils utilisés pour mener à bien le processus de conteneurisation des applications, nous passons maintenant à l'illustration des captures des différentes étapes effectuées.

3.4.1 Mise en place de l'environnement de travail

Au début, nous commencerons notre pratique avec une préparation de l'environnement de travail où nous effectuerons nos tracés d'exécution. Ainsi, nous poursuivrons l'installation de Docker, choisie comme technologie de conteneurisation, suite à une simple démonstration à chaque étape.

Comme une première étape, nous écrirons un script shell, présenté par la figure 3.6, qui comprend toutes les commandes d'installation nécessaires.



```

GNU nano 4.8
#!/bin/bash

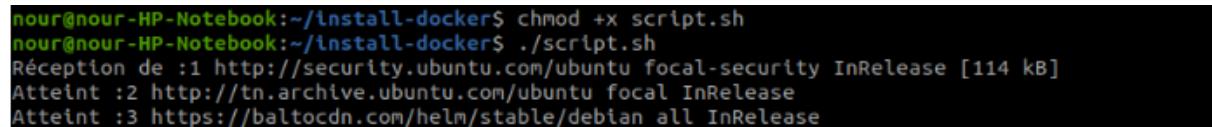
sudo apt-get update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
sudo apt-get update
apt-cache policy docker-ce
sudo apt install docker-ce
sudo systemctl status docker

# Linux post-install
sudo usermod -aG docker $(whoami)

```

FIGURE 3.6 – Script d'installation Docker

Ensuite nous exécuterons ce script à fin d'effectuer l'installation du docker par les commandes qui sont illustrés par la figure 3.7 :



```

nour@nour-HP-Notebook:~/install-docker$ chmod +x script.sh
nour@nour-HP-Notebook:~/install-docker$ ./script.sh
Réception de :1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Atteint :2 http://ttn.archive.ubuntu.com/ubuntu focal InRelease
Atteint :3 https://baltocdn.com/helm/stable/debian all InRelease

```

FIGURE 3.7 – Exécution de script d'installation

Et finalement, nous vérifierons que le service docker a été installé avec succès "Active" sur le local avec la commande « sudo systemctl status docker » qui est présenté par la figure 3.8 :

```
nour@nour-HP-Notebook:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2022-09-03 13:46:08 CET; 1min 10s ago
     TriggeredBy: ● docker.socket
     Docs: https://docs.docker.com
      Main PID: 16387 (dockerd)
        Tasks: 9
       Memory: 39.4M
      CGroup: /system.slice/docker.service
              └─16387 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

13:46:05 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:05.229785640+01:00" level=warning msg="Your kernel does not support CPU re...
13:46:05 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:05.231545364+01:00" level=warning msg="Your kernel does not support cgroup...
13:46:05 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:05.231618574+01:00" level=warning msg="Your kernel does not support cgroup...
13:46:05 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:05.232237308+01:00" level=info msg="Loading containers: start."
13:46:06 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:06.530684358+01:00" level=info msg="Default bridge (dockero) is assigned w...
13:46:06 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:06.821463578+01:00" level=info msg="Loading containers: done."
13:46:06 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:08.133622399+01:00" level=info msg="Docker daemon" commit=a09b842 graphdr...
13:46:06 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:08.134482769+01:00" level=info msg="Daemon has completed initializati...
13:46:06 03 در نور-HP-Notebook systemd[1]: Started Docker Application Container Engine.
13:46:06 03 در نور-HP-Notebook dockerd[16387]: time="2022-09-03T13:46:08.783100003+01:00" level=info msg="API listen on /run/docker.sock"
Lines 1-21/21 (END)
```

FIGURE 3.8 – Statut Docker

Passons maintenant à l'installation du docker-compose qui est responsable à l'orchestration de nos conteneurs lors du déploiement. De même nous écrirons un script, présenté par la figure 3.9, pour l'installer.

```
nour@nour-HP-Notebook:~/Install-docker$ ./install-docker 150x38
GNU nano 4.8
#!/bin/bash

sudo apt update
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

FIGURE 3.9 – Script Docker-Compose

Pour exécuter ce script, nous avons besoin des commandes qui sont présentés dans la figure 3.10 :

```
nour@nour-HP-Notebook:~/Install-docker$ chmod +x docker-compose.sh
nour@nour-HP-Notebook:~/Install-docker$ ./docker-compose.sh
[sudo] Mot de passe de nour :
      % Total    % Received % Xferd  Average Speed   Time   Time   Current
                                     Dload  Upload Total   Spent   Left  Speed
      0      0      0      0      0      0      0 --::-- --::-- --::--   0
  100 11.6M  100 11.6M    0      0  1079k      0  0:00:11  0:00:11 --::-- 1109k
docker-compose version 1.26.0, build d4451659
nour@nour-HP-Notebook:~/Install-docker$
```

FIGURE 3.10 – Installation Docker-Compose

3.4.2 Conteneurisation des micro-services

Comme nous l'avons défini, au début de ce chapitre, notre application a été découpée en 3 micro-services :

- ❖ micro-service front-end sous le nom de 'scope_front'.

- ❖ micro-service api sous le nom 'scope_api'.

- ❖ micro-service backend sous le nom ‘scope_backoffice’.

Pour réussir le concept de conteneurisation de chaque micro-service il faut suivre le processus de fonctionnement Docker (écrire Dockerfile, construire l'image docker et la déployer dans un conteneur).

Donc, dans les parties suivantes, nous poursuivrons les étapes de conteneurisation pour chaque micro-service.

Conteneurisation scope_front

Durant cette partie, nous écrirons le fichier Dockerfile dédié à chaque micro-service, qui comprend toutes les dépendances nécessaires pour construire l'image Docker.

D'où la figure 3.11 présente le Dockerfile adressé au micro-service front : ‘scope_front’.

```
nour@nour-HP-Notebook:~/PFE/scope_front$ cat Dockerfile
FROM node:14.17.0 AS build
LABEL maintainer="Nour Teber <nour.teber@oyez.fr>"
WORKDIR /app
COPY package.json package.json
RUN yarn
COPY .
FROM node:14.17.0
WORKDIR /app
EXPOSE 3000
COPY --from=build /app/
RUN yarn global add serve
CMD ["serve", "-s", "./build", "-l", "3000"]

nour@nour-HP-Notebook:~/PFE/scope_front$
```

FIGURE 3.11 – Dockerfile ‘scope_front’

- ★ **FROM** : Définir l'image docker de base = spécification d'image (Node) avec sa version (v14.17.0) qui se charge lors de la création de notre image docker.
- ★ **LABEL maintainer** : Identificateur en tant que responsable de l'image à créer (Nour Teber dans notre cas).
- ★ **WORKDIR** : Définir le répertoire de travail par défaut des conteneurs (/app).
- ★ **COPY** : Permet de copier des fichiers locaux dans le répertoire des conteneurs (WORKDIR) en garantissant les mêmes configurations.
- ★ **RUN** : Spécifier les commandes pour apporter des modifications à notre image et au conteneur démarré à partir de cette image.

★ **EXPOSE** : Définir les ports d'image à créer (Dans ce micro-service nous utilisons le port 3000 lors de déploiement).

★ **CMD** : Définir la commande qui s'exécutera au démarrage du conteneur. Dans notre Dockerfile, la commande CMD [“serve”, “-s”, “./build”, “-l”, “3000”] : Permet de déployer le micro-service ‘scope_front’ sur le port 3000.

Après l'exécution du fichier ‘Dockerfile’ avec la commande « docker build -t scope_front . » Nous pouvons afficher les images docker obtenues à l'aide de la commande docker ‘sudo docker images’, qui sont présentées dans la figure 3.12 :

```
nour@nour-HP-Notebook:~/PFE$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
scope_api       latest   a5d1ee85d2e2  7 minutes ago  1.02GB
<none>          <none>   36d5524a1e1d  10 minutes ago  1.19GB
scope_front    latest   8bbbd7a0149a  25 minutes ago  1.2GB
<none>          <none>   566e076cc9d4  33 minutes ago  1.66GB
node            14.17.0  9153ee3e2ced  15 months ago  943MB
nour@nour-HP-Notebook:~$
```

FIGURE 3.12 – Docker image ‘scope_front’

Ensuite, nous déployons ce micro-service dans un conteneur spécifique, à l'aide de la commande docker ‘Docker run –name "scope-front" -p 3000 :3000 -d 8bbbd7a0149a’. D'où :

- ✓ **–name** : Définir le nom de conteneur à créer
- ✓ **-p** : Présente les ports des micro-services lors de déploiement
- ✓ **-d** : Option de la commande « docker run » qui permet son exécution en background.
- ✓ **8bbbd7a0149a** : Présente l'identifiant de l'image docker utilisé pour créer ce conteneur.

La figure 3.13 montre les traces de déploiement du conteneur :

```
nour@nour-HP-Notebook:~/PFE/scoped_front$ docker run --name "scope_front" -p 3000:3000 -d b956f5fb7de
778fb2f44c6e41e41245e168124a30864b09194cea6ff58dd55ab5fc297d658
nour@nour-HP-Notebook:~/PFE/scoped_front$ docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
778fb2f44c6e        b956f5fb7de        "docker-entrypoint.s..."   16 seconds ago     Up 5 seconds        0.0.0.0:3000->3000/tcp, :::3000->3000/tcp   scope_front
nour@nour-HP-Notebook:~/PFE/scoped_front$
```

FIGURE 3.13 – Conteneur ‘scope_front’

La figure 3.14 présente l'interface obtenue au déploiement du micro-service ‘scope_front’ en local :

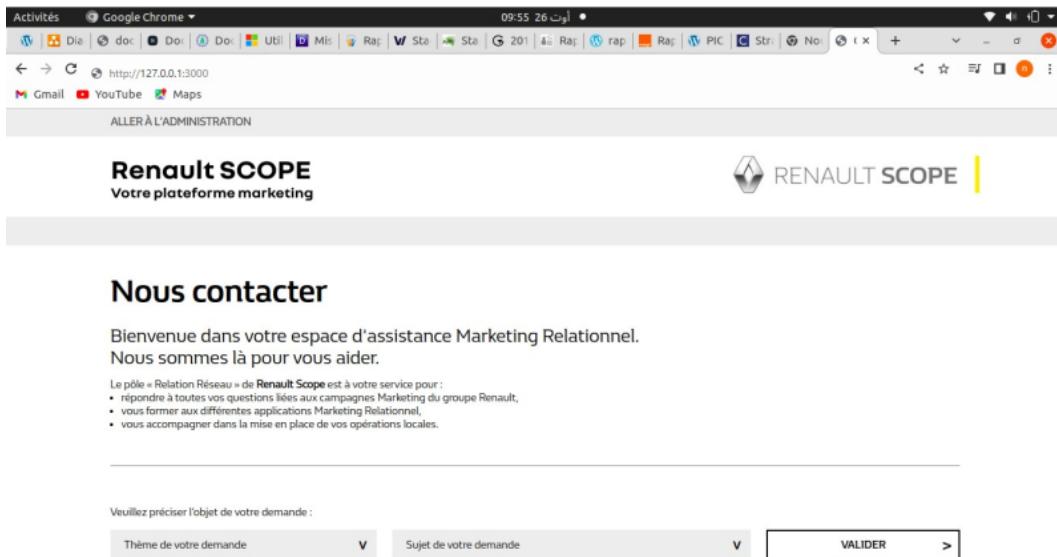


FIGURE 3.14 – L'interface 'scope_front'

Conteneurisation 'scope_api'

De même pour le micro-service 'scope_api', nous écrirons le fichier 'Dockerfile', montré dans la figure 3.15, avec toutes les dépendances nécessaires pour construire l'image docker correspondante. Ce micro-service sera déployer déployé sur le port 4000.

```
nour@nour-HP-Notebook:~/PFE/scope_api$ cat Dockerfile
FROM node:14.17.0 AS next-build
LABEL maintainer="Nour Teber <nour.teber@oyez.fr>"
WORKDIR /app
COPY package.json package.json
RUN yarn
COPY . .
FROM node:14.17.0
WORKDIR /app
EXPOSE 4000
COPY --from=next-build /app/
CMD ["yarn", "start"]
nour@nour-HP-Notebook:~/PFE/scope_api$
```

FIGURE 3.15 – Dockerfile 'scope_api'

Ensuite, nous exécuterons la commande de build pour créer l'image docker qui sera affiché dans la figure 3.16 :

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
scope_api	latest	2033d740d6a0	About an hour ago	1.02GB
<none>	<none>	1573e037d2c6	About an hour ago	1.19GB
scope_front	latest	b956f5fb7de	About an hour ago	1.2GB
<none>	<none>	049a29f1e1fe	2 hours ago	1.65GB
node	14.17.0	9153ee3e2ced	15 months ago	943MB

FIGURE 3.16 – Docker image ‘scope_api’

Et finalement nous terminerons le processus de conteneurisation de ce micro-service par son déploiement dans un conteneur spécifique, illustré dans la figure 3.17 :

nour@nour-HP-Notebook:~/PFE/scope_api\$ docker run --name "scope_api" -p 4000:4000 -d 2033d740d6a0 69bee9d3bfc286cca62468aa6d4db44773984a202d29e7402084e2c5798c61cc
nour@nour-HP-Notebook:~/PFE/scope_api\$ docker ps -a

FIGURE 3.17 – Conteneur ‘scope_api’

La figure 3.18 présente l’interface obtenue du micro-service ‘scope_api’

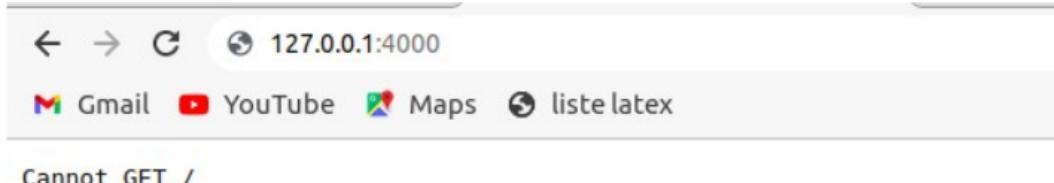
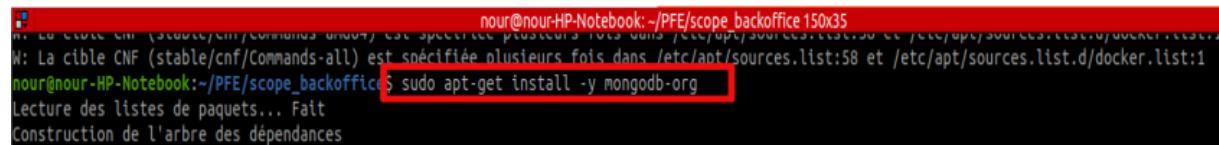


FIGURE 3.18 – L’interface ‘scope_api’

Conteneurisation ‘scope_backoffice’

Passons maintenant à la conteneurisation du troisième micro-service ‘scope_backoffice’, représente la partie backend de notre application et qu’elle doit être liée à la base de données ‘MongoDB’, choisi pour notre solution.

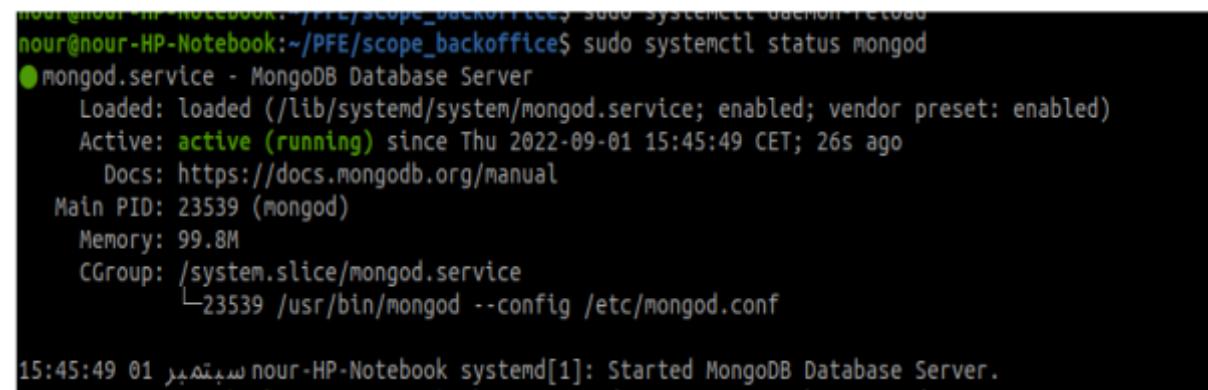
Donc, au début, nous avons besoin d’installer le service ‘MongoDB’, illustré par la figure 3.19 :



```
nour@nour-HP-Notebook:~/PFE/scope_backoffice$ sudo apt-get install -y mongodb-org
W: La cible CNF (/stable/cnf/Commands-all) est spécifiée plusieurs fois dans /etc/apt/sources.list:58 et /etc/apt/sources.list.d/docker.list:1
nour@nour-HP-Notebook:~/PFE/scope_backoffice$ sudo apt-get install -y mongodb-org
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
```

FIGURE 3.19 – Installation du service 'MongoDB'

Ainsi, pour vérifier la réussite de l'installation de ce service, nous tapons la commande Linux présenté par la figure 3.20 :



```
nour@nour-HP-Notebook:~/PFE/scope_backoffice$ sudo systemctl status mongod
● mongod.service - MongoDB Database Server
  Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
  Active: active (running) since Thu 2022-09-01 15:45:49 CET; 26s ago
    Docs: https://docs.mongodb.org/manual
 Main PID: 23539 (mongod)
   Memory: 99.8M
      CGroup: /system.slice/mongod.service
              └─23539 /usr/bin/mongod --config /etc/mongod.conf

15:45:49 01 يوليوز nour-HP-Notebook systemd[1]: Started MongoDB Database Server.
```

FIGURE 3.20 – Statut 'MongoDB'

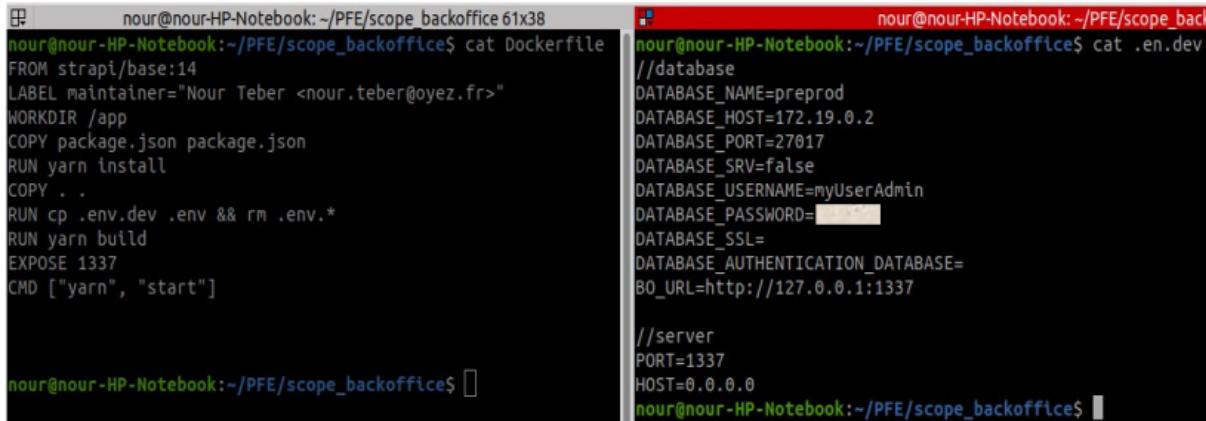
Ensuite, nous créerons un user 'admin', (voir figure 3.21), qui nous permettre de se connecter à notre base de données.



```
...
> use admin
switched to db admin
> db.createUser(
...   {
...     user: "myUserAdmin",
...     pwd: [REDACTED],
...     roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
...   }
...
)
Successfully added user: {
  "user" : "myUserAdmin",
  "roles" : [
    {
      "role" : "userAdminAnyDatabase",
      "db" : "admin"
    }
  ]
}
```

FIGURE 3.21 – Ajout user Mongo

Ensuite, nous écrirons le fichier Dockerfile, montré par la figure 3.22, pour construire l'image docker correspondants, suivi de la configuration de fichier .env qui présente les données nécessaires concernant la connexion entre la base de données 'MongoDB' et ce micro-service.



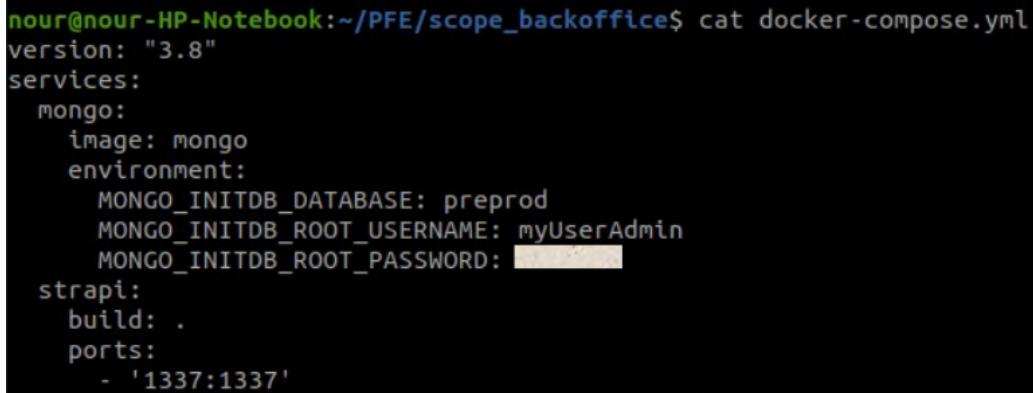
```
nour@nour-HP-Notebook:~/PFE/scope_backoffice$ cat Dockerfile
FROM strapi/base:14
LABEL maintainer="Nour Teber <nour.teber@oyez.fr>"
WORKDIR /app
COPY package.json package.json
RUN yarn install
COPY .
RUN cp .env.dev .env && rm .env.*
RUN yarn build
EXPOSE 1337
CMD ["yarn", "start"]

nour@nour-HP-Notebook:~/PFE/scope_backoffice$ cat .env.dev
//database
DATABASE_NAME=preprod
DATABASE_HOST=172.19.0.2
DATABASE_PORT=27017
DATABASE_SRV=false
DATABASE_USERNAME=myUserAdmin
DATABASE_PASSWORD=[REDACTED]
DATABASE_SSL=
DATABASE_AUTHENTICATION_DATABASE=
BO_URL=http://127.0.0.1:1337

//server
PORT=1337
HOST=0.0.0.0
nour@nour-HP-Notebook:~/PFE/scope_backoffice$
```

FIGURE 3.22 – Dockerfile et .env.dev

A ce moment, nous passerons à préparer le fichier docker-compose.yml – montré par la figure 3.23 - qui comprend les instructions nécessaires pour construire et déployer ce micro-service dans un conteneur docker, en garantissant la connexion entre ce dernier et la base de données.



```
nour@nour-HP-Notebook:~/PFE/scope_backoffice$ cat docker-compose.yml
version: "3.8"
services:
  mongo:
    image: mongo
    environment:
      MONGO_INITDB_DATABASE: preprod
      MONGO_INITDB_ROOT_USERNAME: myUserAdmin
      MONGO_INITDB_ROOT_PASSWORD: [REDACTED]
  strapi:
    build: .
    ports:
      - '1337:1337'
```

FIGURE 3.23 – Docker-compose.yml

La figure 3.24 montre l'interface obtenue du micro-service 'scope_backoffice'

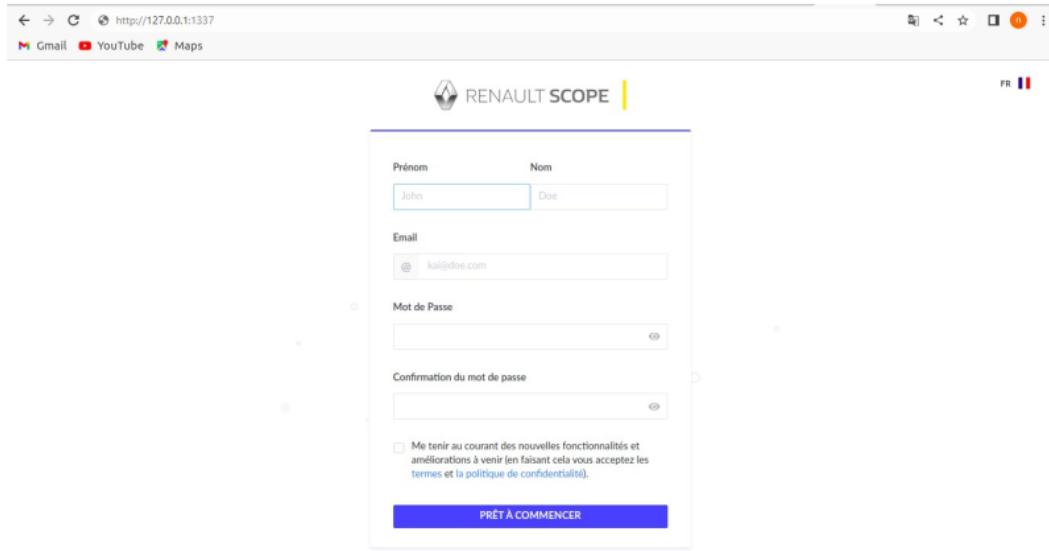


FIGURE 3.24 – L’interface ‘scope_backoffice’

Après avoir conteneurisé chaque micro-service de manière isolée, nous constatons qu’ils ne peuvent pas communiquer entre eux (chaqu’un se connecte avec une adresse IP différente).

Nous avons donc besoin d’un outil d’orchestration, comme Docker-compose, qui permet aux micro-services d’être facilement conteneurisées dans un même environnement.

3.4.3 Docker-compose

Dans cette partie nous définirons la configuration du fichier ‘Yaml’ (docker-compose.yml) – illustrée par la figure 3.25 – qui est responsable pour démarrer tous les conteneurs avec une seule commande, accompagné du déploiement du plateforme ‘Protainer’, qui permet la gestion de conteneurs docker déployé dans notre local.

```
version: "3.8"
services:
  portainer:
    image: portainer/portainer-ce:2.0.0
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - ./portainer-data:/data
    ports:
      - 9000:9000
  scope-front:
    build: ./scope_front
    links:
      - scope_api
    volumes:
      - ./scope_front/:/app
  scope_api:
    build: ./scope_api
    links:
      - scope_backoffice
    volumes:
      - ./scope_api/:/app

  scope_backoffice:
    build: ./scope_backoffice
    environment:
      - ./scope_backoffice/.env.dev
    volumes:
      - ./scope_backoffice/:/app
  mongo:
    image: mongo
    environment:
      MONGO_INITDB_DATABASE: preprod
      MONGO_INITDB_ROOT_USERNAME: myUserAdmin
```

FIGURE 3.25 – Configuration docker-compose.yml

Après l'exécution du script précédent, les conteneurs seront connectés avec la même adresse IP (172.20.0.x), d'où nous avons besoin de servir nos micro-services avec un reverse proxy configuré par le serveur web ‘Nginx’.

La figure 3.26 montre la réussite du fichier d'exécution ‘docker-compose.yaml’ :

```
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
[+] Running 5/5
  ● Network pfe_default          Created
  ● Container pfe-scope-front-1 Started
  ● Container pfe-portaliner-1   Started
  ● Container pfe-scope_api-1    Started
  ● Container pfe-scope_backoffice-strapi-1 Started
  ● Container pfe-scope_backoffice-mongo-1 Started
```

FIGURE 3.26 – Déploiement des conteneurs

Configuration Nginx

La figure 3.27 présente le statut du service 'Nginx' en local :

```
nour@nour-HP-Notebook:~$ systemctl status nginx.service
● nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset:
  Active: active (running) since Fri 2022-09-23 10:24:41 CET; 2h 30min ago
    Docs: man:nginx(8)
   Main PID: 1046 (nginx)
      Tasks: 5 (limit: 4336)
     Memory: 5.0M
        CPU: 0.000 CPU(s) (idle)
       CGroup: /system.slice/nginx.service
               ├─1046 nginx: master process /usr/sbin/nginx -g daemon on; master
               ├─1047 nginx: worker process
               ├─1048 nginx: worker process
               ├─1049 nginx: worker process
               └─1050 nginx: worker process
```

FIGURE 3.27 – Statut 'Nginx'

Ensuite, nous créerons les fichiers de configuration Nginx en spécifiant le nom du serveur(`server_name`) et le `proxy_pass` (qui contient l'adresse IP), après avoir attribué un nom DNS pour les trois micro-services (front, api et backoffice) de notre application dans un fichier de configuration `/etc/hosts`.

La figure 3.28 montre le fichier de configuration du micro-service 'scope_front' :

```

# default server configuration
#
server {
#       listen 80 default_server;
#       listen [::]:80 default_server;

root /var/www/scope;

# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name local.scope-front;

location / {
    proxy_pass http://172.20.0.4:3000;
}
}

```

FIGURE 3.28 – Configuration Nginx ‘scope_front’

La figure 3.29 présente l’interface du micro-service ‘scope_front’ qui a été déployé en local.

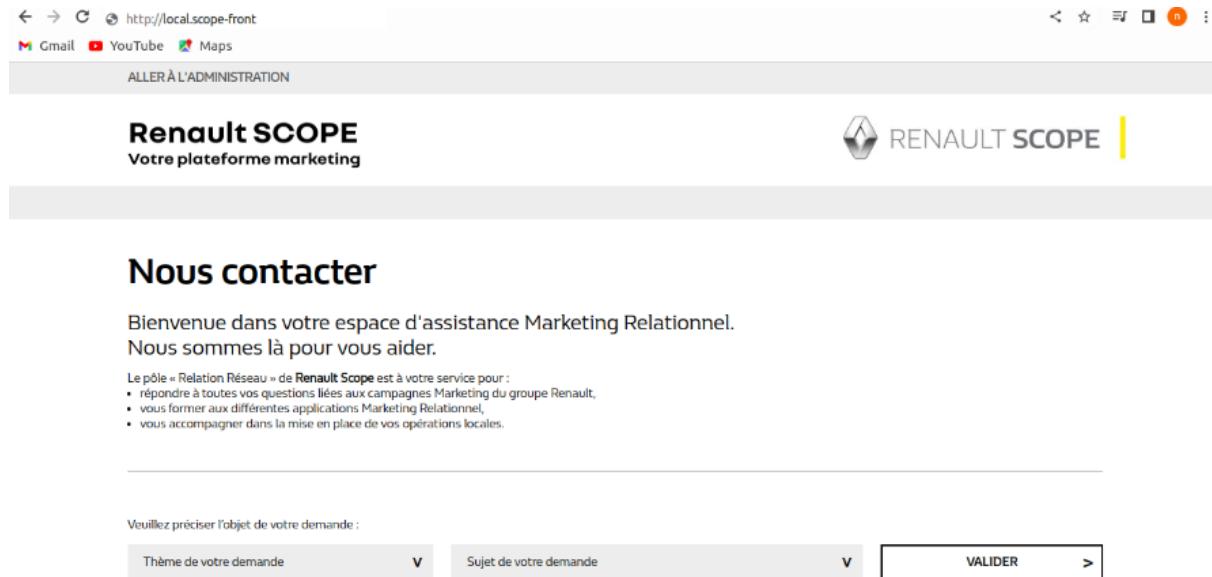


FIGURE 3.29 – Interface ‘local.scope-front’

De même, nous créerons le fichier de configuration ‘Nginx’ du service ‘scope_api’, présenté par la figure 3.30 :

```
root /var/www/scope;

# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name local.scope-api;

location / {
    # First attempt to serve request as file, then
    proxy_pass http://172.20.0.3:4000;
}
```

FIGURE 3.30 – Configuration Nginx ‘scope_api’

D'où la figure 3.31 montre l'interface obtenu de ce service :

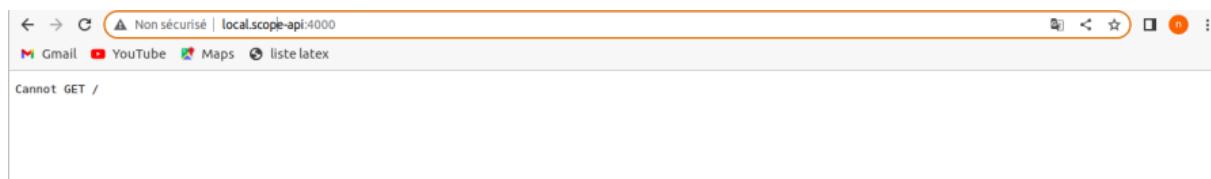


FIGURE 3.31 – Interface ‘local.scope-api’

Et nous finirons par la figure 3.32 qui présente le fichier de configuration du service backoffice :

```
root /var/www/scope;

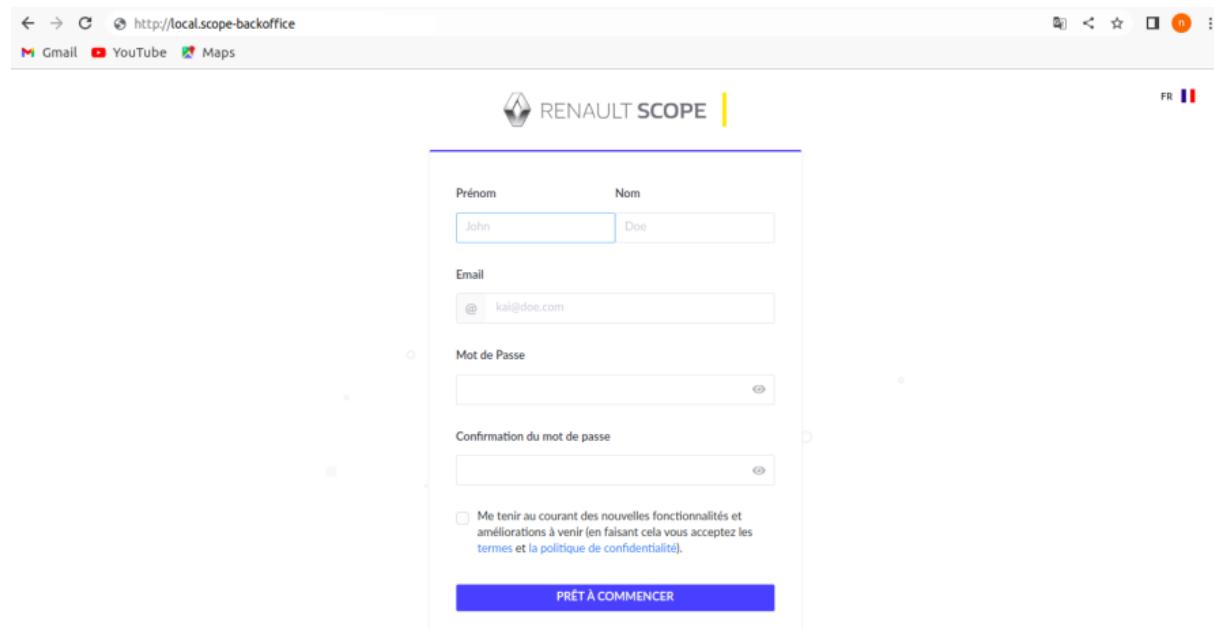
# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name local.scope-backoffice;

location / {
    # First attempt to serve request as file, then
    proxy_pass http://172.20.0.5:1337;
}
```

FIGURE 3.32 – Configuration Nginx ‘scope_backoffice’

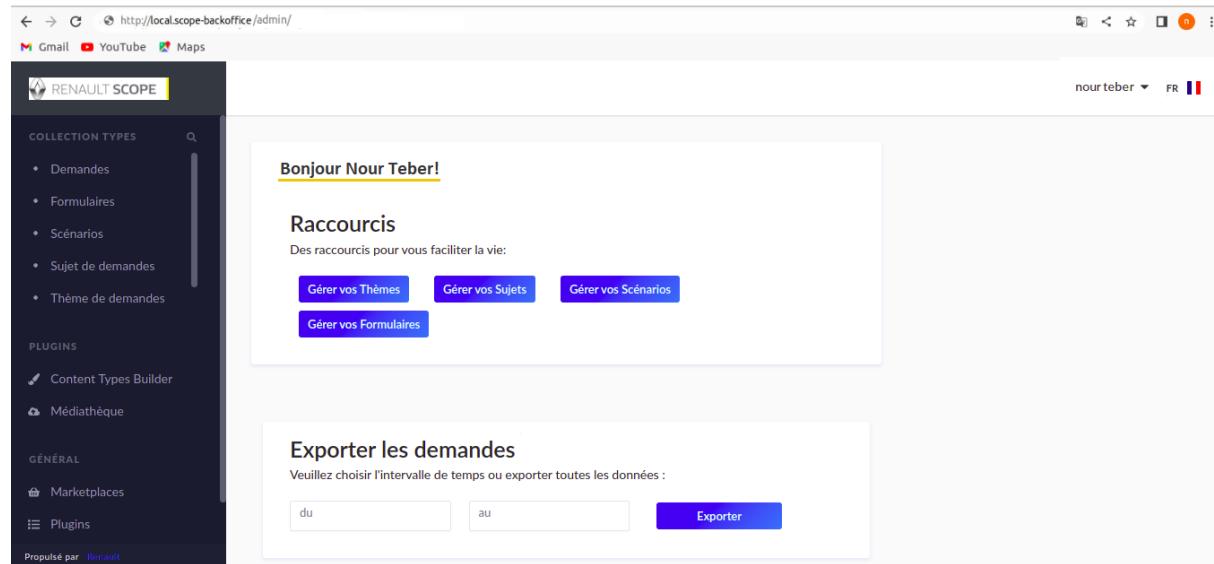
Les deux figures suivantes 3.33 et 3.34 illustrent les interfaces obtenues du service ‘scope_backoffice’



The screenshot shows a web browser window with the URL <http://local.scope-backoffice>. The page title is "RENAULT SCOPE". The main content is a registration form:

- Prénom:** John
- Nom:** Doe
- Email:** kai@doe.com
- Mot de Passe:** (password field)
- Confirmation du mot de passe:** (password confirmation field)
- Checkboxes:** "Me tenir au courant des nouvelles fonctionnalités et améliorations à venir (en faisant cela vous acceptez les termes et la politique de confidentialité)"
- Buttons:** "PRÉT À COMMENCER"

FIGURE 3.33 – Interface login du micro-service ‘scope_backoffice’.



The screenshot shows a web browser window with the URL <http://local.scope-backoffice/admin/>. The page title is "RENAULT SCOPE". The main content area includes:

- Bonjour Nour Teber!**
- Raccourcis**: Des raccourcis pour vous faciliter la vie:
 - Gérer vos Thèmes
 - Gérer vos Sujets
 - Gérer vos Scénarios
 - Gérer vos Formulaires
- Exporter les demandes**: Veillez choisir l'intervalle de temps ou exporter toutes les données :
 - du: [input field]
 - au: [input field]
 - Exporter: [button]

The left sidebar contains navigation links for "COLLECTION TYPES" (Demandes, Formulaires, Scénarios, Sujet de demandes, Thème de demandes), "PLUGINS" (Content Types Builder, Médiathèque), and "GÉNÉRAL" (Marketplaces, Plugins). The footer states "Propulsé par Renault".

FIGURE 3.34 – Interface ‘local.scope-backoffice’

La figure 3.35 montre l'interface du service "Portainer" qui comprend les différentes images dont ils sont construite en local :

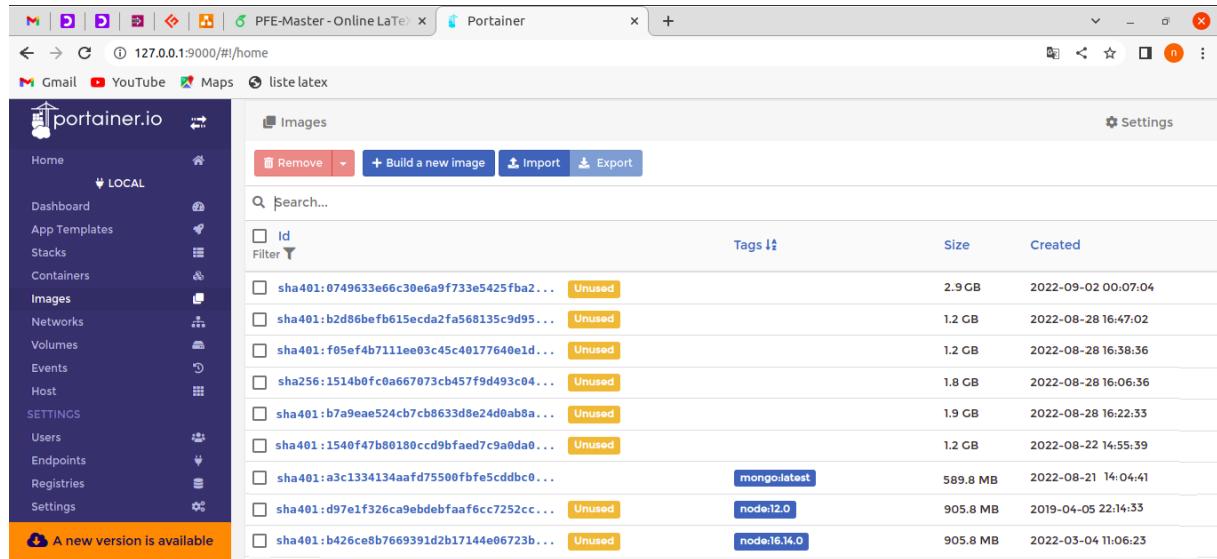


FIGURE 3.35 – Interface 'Portainer'

3.4.4 Registre de conteneurs

Durant cette partie, nous démontrerons le processus d'hébergement des images docker obtenues dans le registre docker privé 'Harbor'.

Préparation de l'environnement

Comme une première pas, nous installerons et implémenterons le service Harbor, le registre docker utilisé à l'hébergement des images docker, à l'aide du docker-compose.

La figure 3.36 illustre le fichier de configuration 'harbor.yml' qui présente le 'server name' adopté pour accéder à la plateforme.

```

hour@nour:~/Notebook/~/harbor$ nano harbor.yml
hour@nour-HP-Notebook:~/harbor$ cat harbor.yml
# Configuration file of Harbor

# The IP address or hostname to access admin UI and registry service.
# DO NOT use localhost or 127.0.0.1, because Harbor needs to be accessed by external clients.
hostname: harbor-scope.local

# http related config
http:
  # port for http, default is 80. If https enabled, this port will redirect to https port
  port: 8080

# https related config
#https:
  # https port for harbor, default is 443
  # port: 443
  # The path of cert and key files for nginx
  #certificate: /your/certificate/path
  #private_key: /your/private/key/path

# # Uncomment following will enable tls communication between all harbor components
#internal_tls:
#  # set enabled to true means internal tls is enabled
#  enabled: true
#  # put your cert and key files on dir
#  dir: /etc/harbor/tls/internal

# Uncomment external_url if you want to enable external proxy
# And when it enabled the hostname will no longer used
# external_url: https://reg.mydomain.com:8433

# The initial password of Harbor admin
# It only works in first time to install harbor
# Remember Change the admin password from UI after launching Harbor.
harbor_admin_password: [REDACTED]

```

FIGURE 3.36 – Fichier de configuration ‘harbor.yml’

Suite à cette configuration, nous lancerons le script d’installation de Harbor, d’où la figure 3.37 illustre la réussite de son déploiement

```

[Step 5]: starting Harbor ...
Creating network "harbor_harbor" with the default driver
Creating harbor-log ... done
Creating registryctl ... done
Creating registry ... done
Creating redis ... done
Creating harbor-db ... done
Creating harbor-portal ... done
Creating harbor-core ... done
Creating harbor-jobservice ... done
Creating nginx ... done
✓ ----Harbor has been installed and started successfully.----
```

FIGURE 3.37 – Installation ‘Harbor’

Hébergement des images docker

Durant cette partie, nous montrerons les captures d’hébergement des images docker de chaque micro-services obtenues dans le registre docker privé ‘Harbor’.

Avant l’exécution d’hébergement des micro-service, nous sommes obligés de se connecter

à notre compte ‘Harbor’.

➤ **Hébergement de l'image ‘scope_front’ :**

Afin de pouvoir créer un dépôt spécifique à ce service, nous poussons son image docker avec la commande ‘docker push’.

La figure 3.38 montre le succès de cette étape.

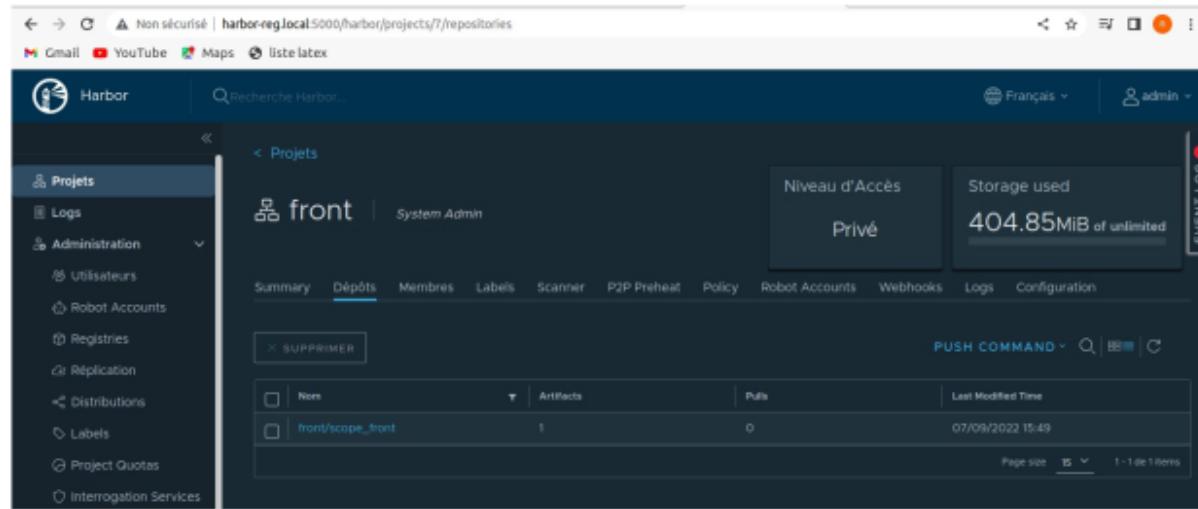


FIGURE 3.38 – Hébergement ‘scope_front’

➤ **Hébergement de l'image ‘scope_api’ :**

De même, nous créeros un dépôt correspond à l'image ‘scope_api’, illustré par la figure 3.39 :

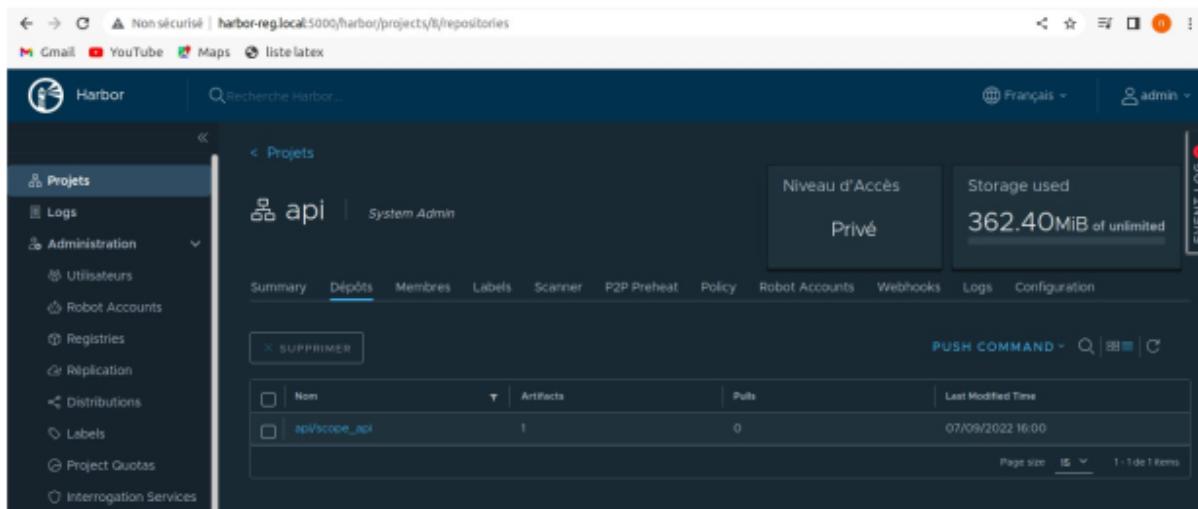


FIGURE 3.39 – Hébergement ‘scope_api’

➤ Hébergement de l'image ‘scope_backoffice’ :

Nous terminerons par l'hébergement de l'image docker ‘scope_backoffice’, montré dans la figure 3.40 :

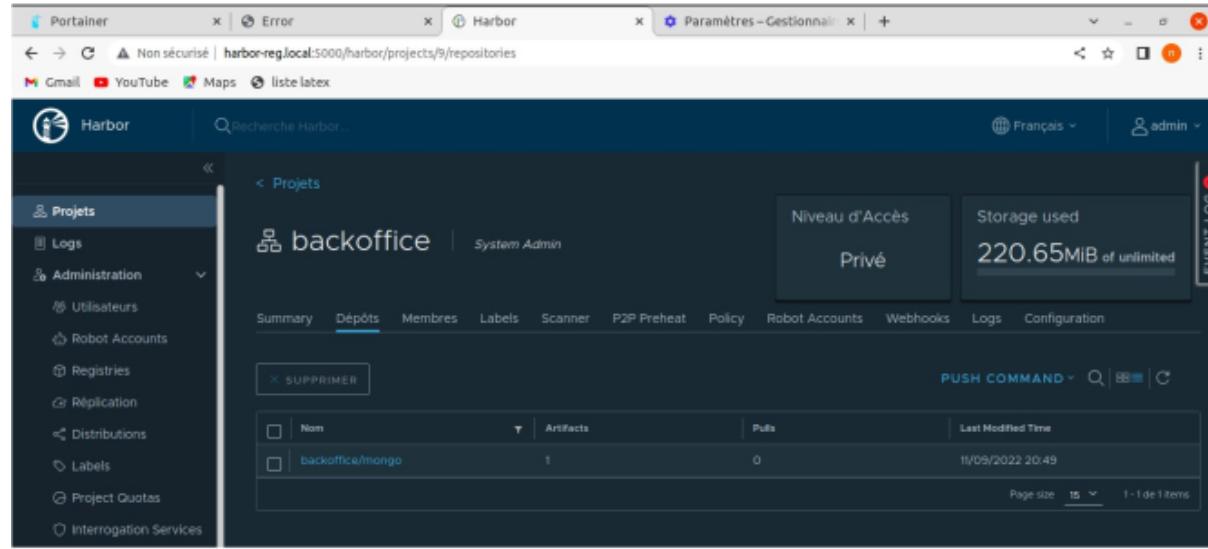


FIGURE 3.40 – Hébergement ‘scope_backoffice’

3.5 Conclusion

Tout au long de ce chapitre, nous avons atteint avec succès le premier objectif de notre projet " Conteneurisation ". Au début, nous avons présenté l'architecture détaillée proposée, puis nous avons résumé les outils adaptés sous la rubrique 'Environnement de travail' et nous avons fini par une liste de captures des tâches à exécuter au cours de cette partie. Dans le chapitre suivant, nous essaierons de voir le déploiement de ces micro-services avec une autre méthode, ainsi que d'expliquer le concept d'automatisation avec la mise en place du pipeline CI/CD.

Chapitre 4

Déploiement des micro-services sur GKE

4.1 Introduction

Dans ce chapitre, nous présenterons le second volet du projet : **Déploiement des micro-services sur GKE**. Commençant par un aperçu de l'architecture proposée, puis nous définirons brièvement les outils et technologies adoptés, et terminerons par une illustration des captures d'écran de l'exécution pratique lors du déploiement de notre application.

L'objectif de ce chapitre est d'automatiser le déploiement d'une application conteneurisée sur « GKE » à l'aide d'un pipeline CI/CD.

4.2 Architecture de déploiement proposée

Comme nous l'avons défini dans les chapitres précédents, notre application se compose de trois micro-services, qui seront conteneurisés et orchestrés par le service SaaS Google Kubernetes Engine (GKE) fourni par le fournisseur de cloud « Google Cloud », tout en garantissant leur performance et leur sécurité, afin que les clients puissent y accéder à tout moment.

Au début, nous avons besoin d'une autorisation au compte 'GCP' pour configurer les services nécessaires au déploiement, tels que le réseau VPC (qui est responsable de la configuration du réseau), Cloud Registry (le registre docker où nous hébergerons nos images docker), IAM (responsable à l'autorisation des utilisateurs à notre application), Storage GCP (service de stockage pour les données sensibles).

Ensuite nous mettrons également en place un cluster Kubernetes 'Googke Kubernetes Engine (GKE)' qui présente l'environnement de déploiement de notre application.

En parallèle, nous préparerons le script du pipeline CI/CD pour chaque micro-service responsable de l'automatisation du déploiement.

Ces différentes étapes seront exécutées sur deux environnements hétérogènes : l'environnement DEV l'environnement Qalif qui ont été définis au chapitre 2 (Architecture générale proposée).

La figure 4.1 montre l'architecture de déploiement proposée :

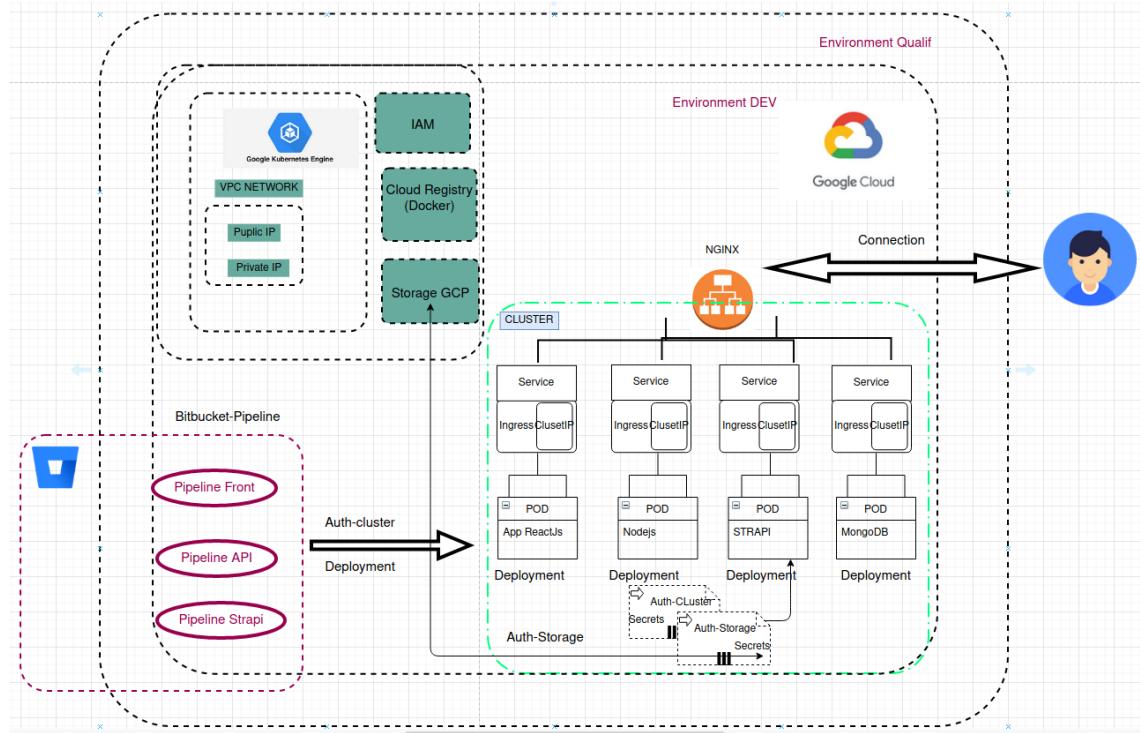


FIGURE 4.1 – Architecture de déploiement proposée

Afin de bien comprendre le but de cette architecture, nous proposons de définir l'environnement de travail en identifiant les outils et technologies adoptés pour atteindre l'objectif de cette section, suivi d'une description détaillée et illustrant également l'exécution des étapes de la forme des captures.

4.3 Environnement du travail

Tout au long de cette partie, nous présenterons les outils nécessaires à l'exécution de ce chapitre.

En commençant, par une introduction de la plateforme cloud « Google Cloud Platform, GCP » en présentant ses services, passant à la documentation du service saas GKE (google kubernetes engine) qui est responsable du déploiement de notre application, et nous finirons par une présentation de l'outil de gestion et d'hébergement du code source ‘Bitbucket’ qui est responsable à l’automatisation à l'aide du pipeline CI/CD ‘bitbucket-pipelines.yml’.

4.3.1 Google Cloud Plateforme

La Google Cloud Platform (GCP) est une convergence de services cloud, proposée par Google, dédiée aux entreprises, aux administrateurs cloud et même aux développeurs pour faciliter leur travail. Donc nous citerons quelques services qui nous avons besoin aux déploiement de notre application comme Cloud Registry, IAM, Storage GCP, VPC Network.



FIGURE 4.2 – Les services de ‘GCP’

VPC Network

Un réseau cloud privé virtuel (VPC) est une version virtuelle d'un réseau physique, mis en œuvre au sein du réseau de production de Google. Il peut configurer automatiquement notre topologie virtuelle en définissant des plages de préfixes pour nos sous-réseaux ainsi que des règles de réseau.

[?]

Ce réseau virtuel propose plusieurs fonctionnalités comme :

- ▶ Permet la connectivité aux instances de machines virtuelles (VM) voire les nœuds d'un clusters Google Kubernetes Engine (GKE).
- ▶ Propose des systèmes d'équilibrage de charge TCP/UDP interne natif et de proxy pour l'équilibrage de charge HTTP(S) interne.
- ▶ Il peut connecter aux réseaux sur site à l'aide de tunnels Cloud VPN et de rattachements d'interconnexion Cloud.
- ▶ Permet de distribuer le trafic des équilibreurs de charge externes Google Cloud aux backends.

IAM (Identity and Access Management)

Permet aux administrateurs d'accorder les autorisations nécessaires aux utilisateurs travaillant sur des ressources spécifiques. Cette solution permet d'adopter le principe de

sécurité du moindre privilège pour faciliter le contrôle des accès aux ressources Google Cloud (ainsi les projets).

En résumé, le principe de l’IAM est de permettre de contrôler qui (identité) a quelles autorisations d’accès (rôles) à quelles ressources, en attribuant un ou plusieurs rôles spécifiques à un membre du projet, et ainsi d’accorder à cette identité certaines autorisations. [?]

Cloud Registry

Avec ce registre, les équipes disposent d’un outil centralisé qui lui permet de gérer leurs images Docker, d’analyser les failles et de définir avec une grande précision les droits d'accès de chaque utilisateur.

Ainsi, les intégrations CI/CD existantes vous offrent la possibilité de configurer des pipelines Docker entièrement automatisés afin de recevoir rapidement des retours. [?]

Storage GCP

Google Cloud Storage est le service de stockage d’objets proposé par Google Cloud. Il fournit d’excellentes fonctionnalités prêtes à l’emploi telles que la gestion des versions d’objets ou des autorisations granulaires (par objet ou compartiment), ce qui peut faciliter le développement et aider à réduire les frais généraux opérationnels. Google Cloud Storage sert de base à plusieurs services différents.

[?]

Google Kubernetes Engine (GKE)

Google Kubernetes Engine (GKE) est un environnement configuré, qui permet le déploiement, la gestion, voire l’évolution des applications conteneurisé grâce aux services fournies par l’infrastructure Google. Il regroupe plusieurs machines (nœuds), pour former un cluster de conteneurs.

Les clusters Google Kubernetes Engine (GKE) sont basés sur le système de gestion de cluster open source K8. Son principe est de permettre l’interaction entre l’administrateur et le cluster de conteneurs. À l’aide des commandes et des ressources K8s, qui facilitent le ménagement des applications, en permet l’exécution des tâches administratives, définir des politiques et surveiller l’état des solutions déployées.

[?]

Google Kubernetes Engine (GKE) repose sur des plusieurs principes et avantages tels :

- ▶ **Facilite la création et la gestion et La création des clusters :** Fournit un interface graphique pour faciliter la gestion des cluster, dont il existe deux méthodes différentes pour créer et gérer ce dernier :
 - ★ GKE Autopilote : GGérer les noeuds avec une configuration manuelle minimale, d'où ces derniers seront créés, provisionnés et maintenus automatiquement.
 - ★ GKE standard : Gestion des noeuds manuellement. D'où nous sommes obligés d'intervenir à la configuration des données du cluster et des noeuds.
- ▶ **L'équilibrage de charge (Load Balancing) :** Google Cloud propose l'équilibrage de charge côté serveur qui est responsable à la distribution du trafic entrant

sur plusieurs instances de machines virtuelles (VM)(nœuds dans un cluster GKE). Ce service garantit la redondance et la haute disponibilité de ses composants. D'où, si un composant d'équilibrage de charge échoue, il est redémarré ou automatiquement et immédiatement remplacé.

[?]

L'équilibrage de charge offre les avantages suivants :

- ★ Évoluer les applications
- ★ Supporter un trafic dense
- ★ Acheminer le trafic vers la machine virtuelle la plus proche (nœuds)
- ★ Déetecter et supprimer automatiquement les instances de machines virtuelles non opérationnelles à l'aide des vérifications d'état. Dont, les instances redevenues opérationnelles sont automatiquement rajoutées.

► **L'auto-scaling** : Utilisé à l'ajout ou suppression automatiquement des nœuds. Cela permet aux applications de fonctionner de façon optimale en réduire le coût d'allocation des nouvelles ressources. Il existe trois types d'auto-scaling :

★ Au niveau des workloads (pods) :

- HPA (Horizontal Pod Autoscaling) : Permet d'ajouter ou supprimer des pods en fonction des ressources (CPU, mémoire RAM)
- VPA (Autoscaling de pods verticaux) : Permet de dimensionner les pods en redimensionner ces ressources (allocation plus, ou moins, de ressources CPU et mémoire aux pods existants)

★ Au niveau d'infrastructure (nœuds) :

(Cluster Autoscaler) : Ajouter ou supprimer des nœuds en fonction des pods (Charge de travail).

► **L'automatisation de mise à niveau** : L'automatisation de mise à niveau permet de mettre à jours les nœuds à chaque modification. Du coup, lorsque le déploiement d'une application échoue, elle se redéploie automatiquement. Ce service sera activé par défaut lors de la création d'un cluster ou d'un pool de nœuds.

4.3.2 Bitbucket

Bitbucket est un service SaaS (Software as a Service), créé en 2008, rebaptisé depuis Bitbucket Cloud, édité par la société australienne Atlassian.

C'est un outil de gestion de versions logicielles basé sur les systèmes open source de contrôle de révisions Git et Mercurial., il englobe également un espace de gestion de projet, un gestionnaire de planification, un environnement de collaboration autour du code source, mais aussi des fonctions de test et de CI/CD, pour intégration et livraison continu.[?]



FIGURE 4.3 – Bitbucket

Bitbucket Pipeline

Bitbucket Pipelines est un service CI/CD intégré à la plateforme Bitbucket. Il permet de créer, de tester et même de déployer automatiquement le code en fonction d'un fichier de configuration dans le référentiel.



FIGURE 4.4 – Bitbucket pipeline

4.4 Etape de déploiement des micro-services

Dans cette partie, nous illustrerons les différentes pratiques à exécuter dans notre projet. Ces étapes seront divisées en quatre parties : une première partie dédiée au provisioningnement de l'environnement de travail, une deuxième partie montre l'avancement du processus de déploiement configuré par le pipeline ci/cd, une troisième partie sera allouée pour justifier le succès de déploiement sur le cluster GKE et une quatrième partie qui présente les différentes interfaces de notre application.

4.4.1 Provisionnement de l'environnement de travail

Nous consacrons cette partie à la création de notre projet dans la plateforme cloud GCP, le provisionnement du cluster GKE ainsi de fournir les ressources nécessaires au déploiement.

Création du projet

Comme une première étape nous avons besoins de créer notre projet intitulé ‘scope’ sur la plateforme ‘GCP’, illustrée dans la figure 4.5 :

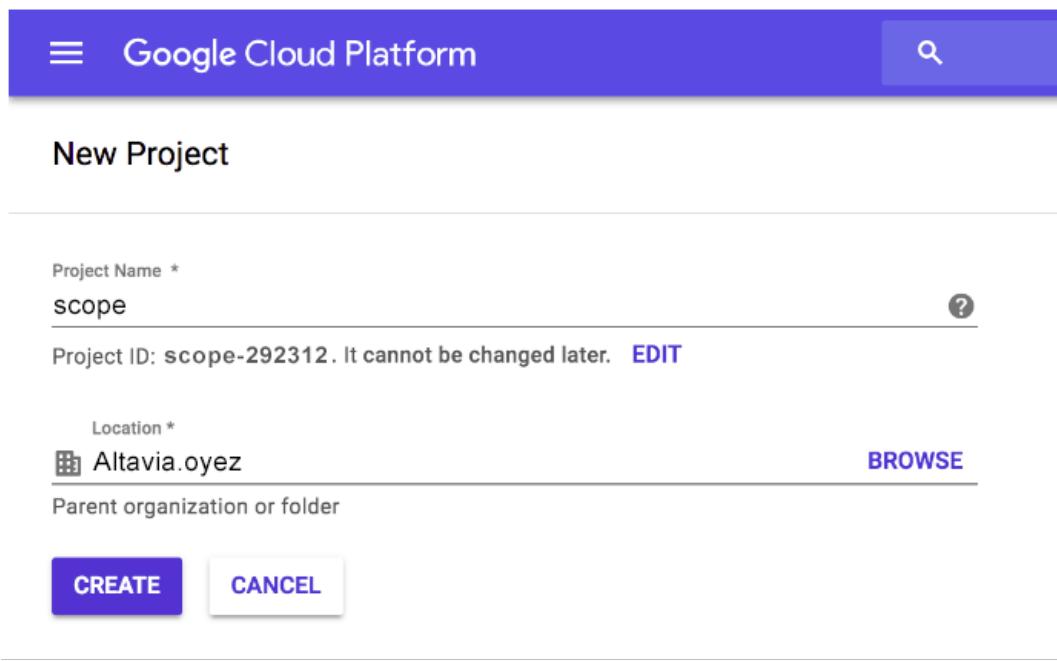


FIGURE 4.5 – Crédit d'un nouveau projet

Création d'un compte service (IAM)

Ensuite, nous avons besoin d'une autorisation pour accéder à ce projet, qui sera représenté sous forme d'un service account .

La figure 4.6 présente le compte service créé :test

The screenshot shows the Google Cloud Platform interface for managing service accounts. The left sidebar includes options like IAM, Identity & Organization, Policy Troubleshooter, Policy Analyzer, Organization Policies, Service Accounts (selected), Workload Identity Federation, Labels, Tags, Settings, Manage Resources, and Release Notes. The main content area displays a table titled "Service accounts for project 'scope'". The table has columns for Email, Status, Name, Description, Key ID, and Actions. It lists four service accounts:

Email	Status	Name	Description	Key ID	Actions
ci-cd-560@scope-292312.iam.gserviceaccount.com	✓	CI/CD		4c3e346f8f9d103ed705df59c2aae0b43beda7d34588bb8ff23d724a219d826f44b476fa1a3e607fade01125c956e9a6ddc94575334effc7d46960f77194b8e99e113df821e502cc	⋮
707724378796-compute@developer.gserviceaccount.com	✓	Compute Engine default service account	No keys		⋮
test-250@scope-292312.iam.gserviceaccount.com	✓	test		805b743b465451409aab0f5dab464da9a4	⋮
uploads-scope@scope-292312.iam.gserviceaccount.com	✓	uploads-scope		d39ca10d0673b537a1ffb7cd8659cda701af	⋮

FIGURE 4.6 – Création d'un compte service

Création d'un réseau VPC

Passons à la création du réseau VPC -montrer dans la figure 4.7, qui sera alloué en tant qu'adresse IP interne dans le cluster 'GKE'.

The screenshot shows the Google Cloud Platform interface for managing VPC networks. The left sidebar includes options like VPC networks (selected), Compute Engine, Network Services, and Monitoring. The main content area displays a table titled "VPC networks". The table has columns for Name, Region, Subnets, MTU, Mode, Internal IP ranges, External IP ranges, Secondary IPv4 ranges, Gateways, Firewall Rules, and Global dynamic. A single row is shown for the 'default' network in the us-central1 region.

Name	Region	Subnets	MTU	Mode	Internal IP ranges	External IP ranges	Secondary IPv4 ranges	Gateways	Firewall Rules	Global dynamic
default	us-central1	default	35 1460	Auto	None			10.128.0.1	4	Off

FIGURE 4.7 – Création du réseau VPC

Cloud registry

De plus, nous créerons un registre docker pour chaque micro-service afin d'héberger les images docker obtenues à chaque création. La figure 4.8 . présente ces registres.

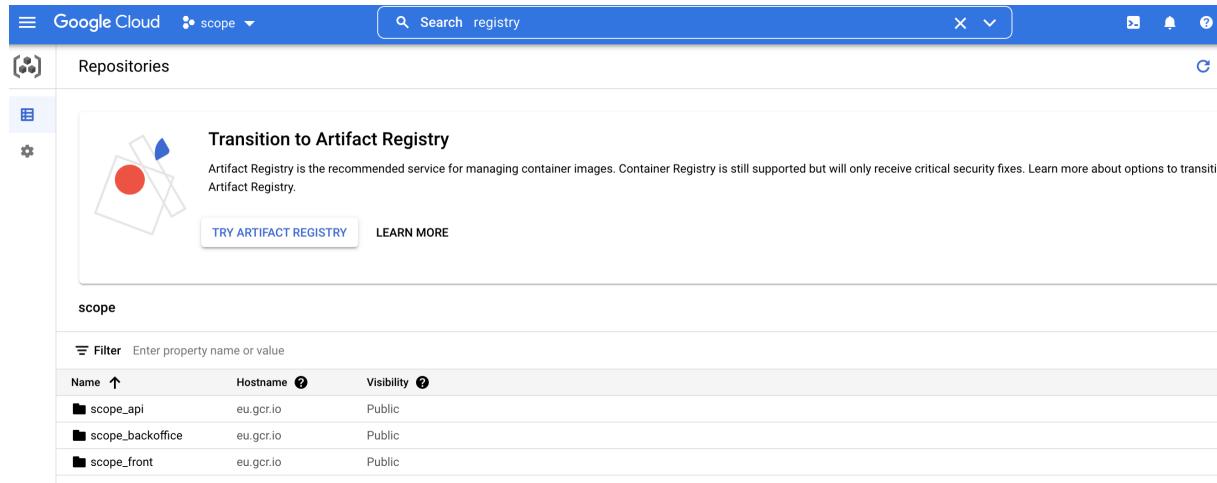


FIGURE 4.8 – Création du registre docker

Création Cluster GKE

Comme nous l'avons défini dans les parties précédentes, un cluster est un groupe de machines virtuelles (nœuds) qui fonctionnent ensemble en mode haute disponibilité. Il s'agit d'une infrastructure fournie par Kubernetes pour la gestion des pods et des nœuds. Ainsi, la figure 4.9 montre la création du cluster GKE "scope-recette" avec les données nécessaires, telles que le type d'allocation (régional/zonal), sa version, le nombre de nœuds à créer ainsi sa mise en réseau.

The screenshot shows the Google Cloud Platform interface for creating a new GKE cluster named 'scope-recette'. The left sidebar has 'Clusters' selected. The main panel shows the cluster details and configuration. The 'Cluster basics' section includes:

- Name: scope-recette
- Location type: Zonal
- Control plane zone: europe-west1-c
- Default node zones: europe-west1-c
- Release channel: None
- Version: 1.21.12-gke.1500
- Total size: 1
- Endpoint: [Show cluster certificate](#)

The right panel shows the configuration for the cluster:

- Automation** section:
 - Maintenance window: Any time
 - Maintenance exclusions: None
 - Notifications: Disabled
 - Vertical Pod Autoscaling: Disabled
 - Node auto-provisioning: Disabled
 - Autoscaling profile: Balanced
- Networking** section:
 - Private cluster: Disabled
 - Network: scope-1pc
 - Subnet: scope-1pc
 - VPC-native traffic routing: Disabled
 - Cluster pod address range (default): 10.112.0.0/14
 - Service address range: 10.115.240.0/20
 - Intranode visibility: Disabled
 - NodeLocal DNSCache: Disabled
 - HTTP Load Balancing: Enabled

FIGURE 4.9 – Création cluster GKE ‘scope-recette’

Dès que le cluster GKE est créé, une ou plusieurs nœuds seront lancées automatiquement en fonction du nombre de CPUs et de mémoire que nous avons définis lors de la

création du cluster, dans notre cas nous en avons 3 nœuds. La figure 4.10 . présenter les nœuds créés.

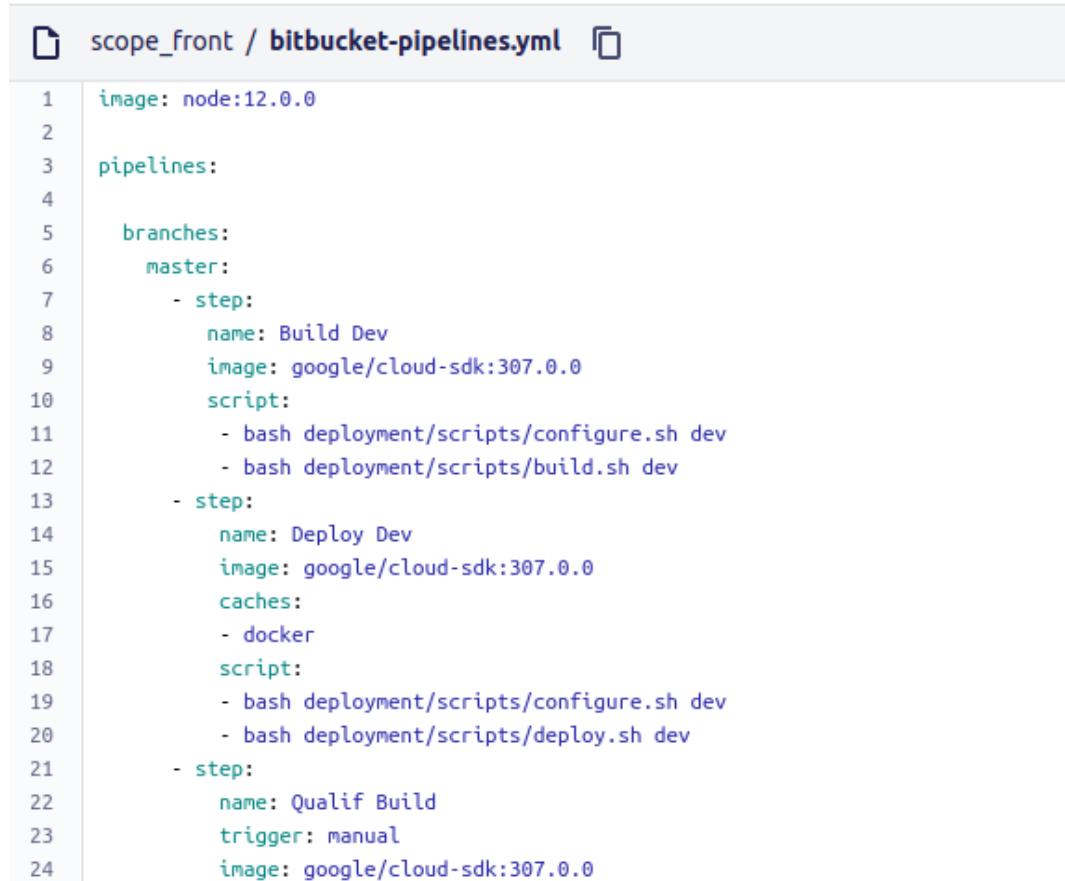
Name	Status	CPU requested	CPU allocatable	Memory requested	Memory allocatable	Storage requested	Storage allocatable
gke-scope-recette-default-pool-532dd674-fazt	Ready	883 mCPU	940 mCPU	656.41 MB	2.95 GB	0 B	0 B
gke-scope-recette-default-pool-532dd674-waf7	Ready	251 mCPU	940 mCPU	372.24 MB	2.95 GB	0 B	0 B
gke-scope-recette-default-pool-532dd674-wmm3	Ready	763 mCPU	940 mCPU	670.6 MB	2.95 GB	0 B	0 B

FIGURE 4.10 – Cr éation des noeuds

Dès que le provisionnement du cluster GKE « scope-recette » sera terminé, nous passerons à la création des fichiers correspondant au déploiement des micro-services en suivant le processus d'intégration continue/déploiement continu (CI/CD).

4.4.2 Pipeline CI/CD

L'objectif de cette partie est de mettre en place le pipeline ci/cd afin d'automatiser le déploiement de l'application, d'où la figure 4.11 présente le fichier ‘bitbucket-pipeline.yml’.



```
1 image: node:12.0.0
2
3 pipelines:
4
5   branches:
6     master:
7       - step:
8         name: Build Dev
9         image: google/cloud-sdk:307.0.0
10        script:
11          - bash deployment/scripts/configure.sh dev
12          - bash deployment/scripts/build.sh dev
13       - step:
14         name: Deploy Dev
15         image: google/cloud-sdk:307.0.0
16         caches:
17           - docker
18         script:
19           - bash deployment/scripts/configure.sh dev
20           - bash deployment/scripts/deploy.sh dev
21       - step:
22         name: Qualif Build
23         trigger: manual
24         image: google/cloud-sdk:307.0.0
```

FIGURE 4.11 – Pipeline ‘scope_front’

Pour réussir l'exécution de ce fichier nous sommes en besoins de déterminer les configurations nécessaires.

Création d'un répertoire des variables

Commencant, par l'ajout des variables dans le répertoire du projet situé dans la plateforme bitbucket.

La figure 4.12 présente le répertoire des variables du ‘Bitbucket’.

Name	Value	Secured
GCLLOUD_API_KEYFILE	<input checked="" type="checkbox"/> Secured
BITBUCKET_REPO_SLUG	<input checked="" type="checkbox"/> Secured
GCLLOUD_PROJECT	<input checked="" type="checkbox"/> Secured
GCLLOUD_ZONE	<input checked="" type="checkbox"/> Secured

FIGURE 4.12 – Répertoire des variables du ‘Bitbucket’

Intégration continue

La configuration de cette phase sera devisée en deux scripts shell : ‘config.sh’ et ‘build.sh’ La figure 4.13 montre le première script ‘config.sh’ dédié au déploiement de micro-service fornt ‘scope_front’, d’où il comprend les instructions nécessaires tel que :

- L’exportation des donnés situé dans le répertoire des variable bitbuket.
- L’authentification à notre compte GCP à l’aide de la clé gérée par le service account.
- La mention du nom du projet prédéfini, ainsi sa position (zone).

```
GNU nano 4.8                                     configure.sh
#!/bin/bash

export GCLLOUD_PROJECT=scope
export GCLLOUD_ZONE=europe-west1-c
export GCLLOUD_API_KEYFILE=${GCLLOUD_API_KEYFILE_DEV}

kubectl version --client

# Authenticating with the service account key file
echo ${GCLLOUD_API_KEYFILE} | base64 --decode --ignore-garbage > ~/cicd-cluster-dev.json
gcloud auth activate-service-account --key-file ~/cicd-cluster-dev.json
# Linking to the Google Cloud project
gcloud config set project ${GCLLOUD_PROJECT}
gcloud config set compute/zone ${GCLLOUD_ZONE}
```

FIGURE 4.13 – Script ‘config.sh’

Déploiement Continu

Nous terminerons le processus de déploiement avec la phase de déploiement continu. Et pour réussir cette phase, nous devons installer ‘Helm’ qui nous aide avec ses templates

(Chart) à définir, installer et mettre à niveau les applications avec Kubernetes. La figure 4.14 montre les instructions d’installation du ‘Helm’ configurés dans le script ‘config.sh’.

```
# Helm
HELM_VERSION=v3.3.4
curl -o https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
bash ./get-helm-3
helm version --client
```

FIGURE 4.14 – Script ‘config.sh’

Ainsi que, la figure 4.15 présente l’arborescence du chart helm correspond au micro-service ‘scope-front’. D’où ces fichiers sont prédéfinis comme suite :

- ⇒ **Le fichier Chart.yaml** : c’est le fichier des métadonnées sur la chart (comprend les informations nécessaires : nom du l’utilisateur, l'url du dépôt bitbucket, le nom du chart, etc.).
- ⇒ **Le fichier values.yaml** : C’est un fichier qui contient les valeurs de configuration par défaut d’une chart.
- ⇒ **Les deux fichiers dev-values.yaml et qa-values.yaml** : ce sont des fichiers qui héritent les mêmes configurations du fichier ‘values.yaml’ accompagnés par les modifications nécessaires.
- ⇒ **Les deux fichiers ‘nginx.dev.conf’ et ‘nginx.qa.conf’** : Héritent les configuration du fichier nginx.conf, fichier par défaut du chart helm, accompagnés par des modifications selon nos besoins.
- ⇒ **Le répertoire templates** : Contient les fichiers utilisés à la configuration kubernetes (les manifest du kubernetes) tel que :
 - ★ configmap.yaml : Un manifeste de base pour gérer les données du configmap (un objet de stockage des données de configuration).
 - ★ deployment.yaml : Un manifeste de base pour créer un déploiement Kubernetes
 - ★ ingress.yaml : présente la configuration des accès externes aux services dans un cluster (trafic HTTP).
 - ★ service.yaml : Un manifeste de base pour créer un point de terminaison de service pour votre déploiement.
 - ★ hpa.yaml : Fichier d’auto-scaling

```
nour@nour-HP-Notebook:~/PFE/scope_front/deployment$ tree scope-front/
scope-front/
├── Chart.yaml
├── dev-values.yaml
├── nginx.conf
├── nginx.dev.conf
├── nginx.qa.conf
├── qa-values.yaml
└── templates
    ├── configmap.yaml
    ├── deployment.yaml
    ├── _helpers.tpl
    ├── hpa.yaml
    ├── ingress.yaml
    └── NOTES.txt
    └── service.yaml
    └── values.yaml

1 directory, 14 files
```

FIGURE 4.15 – Arborescence Helm chart ‘scope_front’

La figure 4.16 présente l’instruction utilisés à la phase de déploiement, qui comprend l’authentification au cluster, ainsi la commande d’installation du chart helm ‘scope_front’.

```
#!/bin/bash

# connect to cluster
export GCLOUD_PROJECT=scope-292312
export GCLOUD_ZONE=europe-west1-c
export KUBERNETES_CLUSTER=scope-recette

gcloud container clusters get-credentials ${KUBERNETES_CLUSTER} --zone ${GCLOUD_ZONE} --project ${GCLOUD_PROJECT}

export IMAGE_NAME=scope-front
export IMAGE_TAG=${ENV}-${BITBUCKET_COMMIT}

echo "Deploying version: ${IMAGE_TAG}"
helm upgrade --install ${IMAGE_NAME} deployment/${IMAGE_NAME} --values deployment/${IMAGE_NAME}/${ENV}-values.yaml --set image.tag=${IMAGE_TAG}
```

FIGURE 4.16 – Instruction de déploiement ‘scope_front’

Exécution pipeline

Dès que nous pousserons les projets locaux de chaque micro-service dans son référentiel situé dans la plateforme ‘Bitbucket’ à l’aide de la commande git “git push”, l’exécution des pipelines se déclenche automatiquement. La figure 4.17 montre le succès d’exécution de notre pipeline de micro-services front.

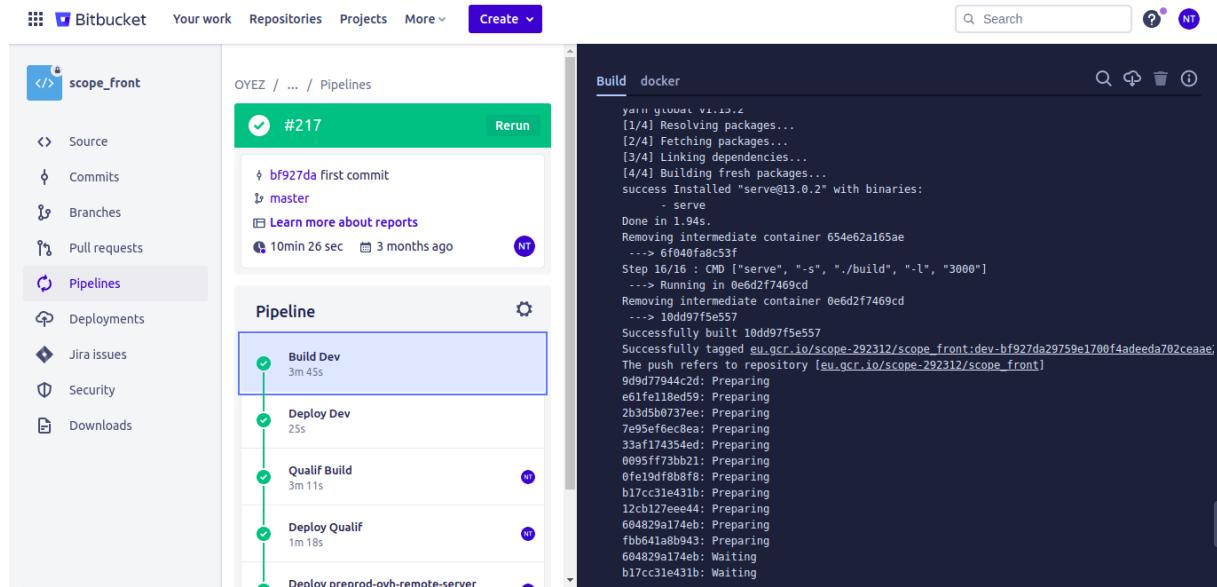


FIGURE 4.17 – Pipeline ‘scope_front’

De même, nous poursuivrons les étapes précédentes pour déployer les micro-services ‘scope_api’ et ‘scope_backoffice’.

D'où la figure 4.18 présente l'exécution du pipeline ‘scope_api’.

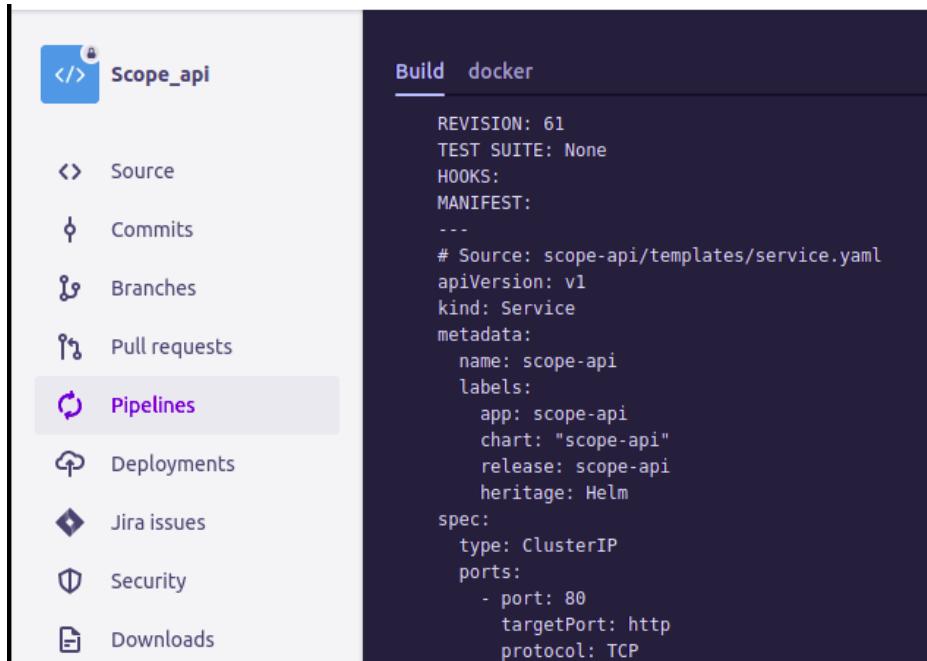


FIGURE 4.18 – Pipeline ‘scope_api’

Ainsi, la figure 4.19. montre le succès d'exécution du pipeline ‘scope_backoffice’.

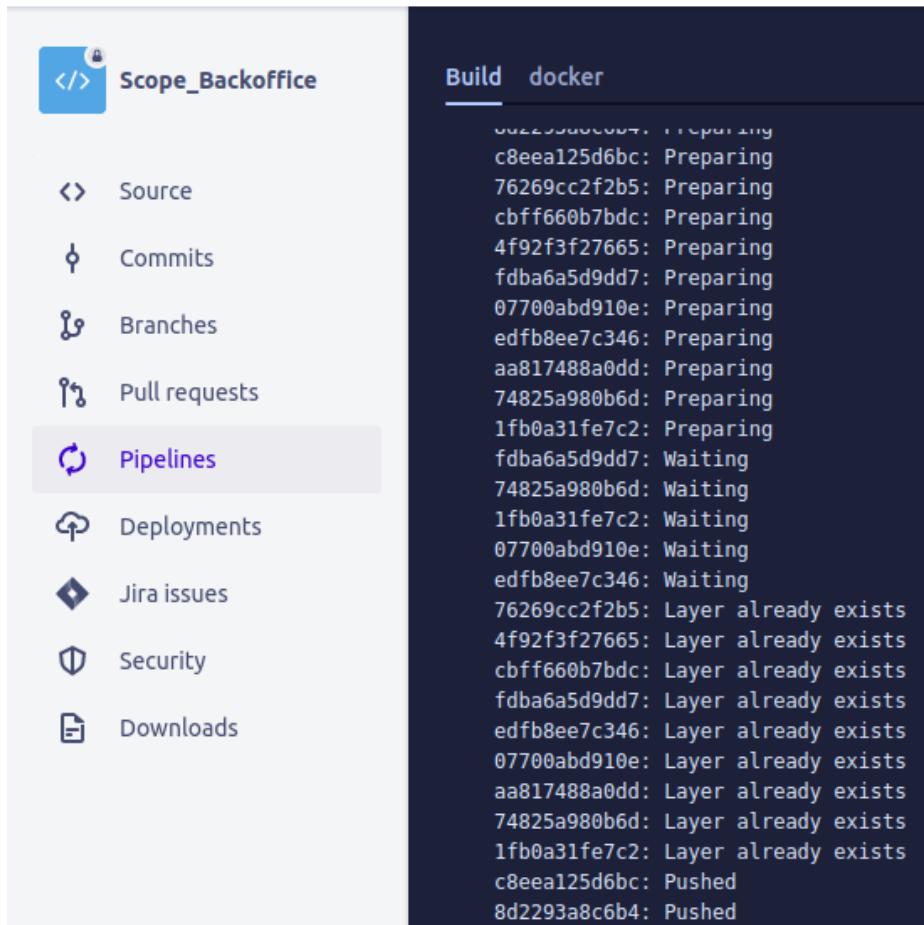


FIGURE 4.19 – Pipeline ‘scope_backoffice’

4.4.3 Déploiement de l’application sur le cluster GKE

Hébergement des images docker

Après exécution des pipelines, l’image docker de chaque micro-service sera hébergée dans son registre cloud. La figure 4.20 présente le registre cloud du micro-service ‘scope_front’ :

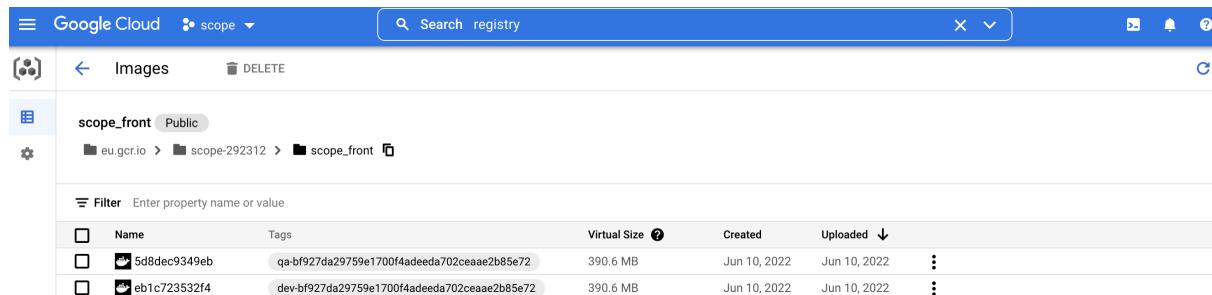


FIGURE 4.20 – Le registre cloud du micro-service ‘scope_front’

La figure 4.21 montre le registre cloud dédié au micro-service ‘scope_api’

Name	Tags	Virtual Size
38997668fb57	q3-585b876ab99196ac7a578c0311c6b5dfcc1dbd3	349.3 MB
73a9047326ca	dev-585b876ab99196ac7a578c0311c6b5dfcc1dbd3	349.3 MB

FIGURE 4.21 – Le registre cloud dédié au micro-service ‘scope_api’

De même, la figure 4.22 présente l’allocation des images docker du micro_service ‘scope_backoffice’ dans son registre docker.

Name	Tags	Virtual Size
327b72299af0	qa-cf17474f20c6be0ee43a8bb3484ad67f78ec0fc31	447.7 MB
5e604c430013	dev-c07474f20c6be0ee43a8bb3484ad67f78ec0fc31	447.7 MB

FIGURE 4.22 – Allocation des images docker du micro-service ‘scope_backoffice’

Création des pods

Dès que l’exécution des pipelines (des micro-services) se termine, des pods (Deployment objects) seront créer. Dans notre cas, nous avons créé 6 pods, présenté dans la figure 4.23 :

- ⇒ scope_front : Présente le micro-service front-end de notre application.
- ⇒ scope_api : Présente le micro-service api de notre application.
- ⇒ scope_backoffice : C'est le micro-service backend de notre application.
- ⇒ mongodb_scope : La base de données MongoDB.
- ⇒ nginxingresscontroller : Permet de Configure le fonctionnement du LoadBalancer HTTP.
- ⇒ nginx_nginx_default_backend : Permet de gérer les accès entre les pods.

The screenshot shows the Google Cloud Workloads interface. At the top, there are buttons for Refresh, Deploy, and Delete. Below that is a search bar and a toolbar with icons for Operations and Help Assistant. The main area displays a table of workloads. The columns are: Name, Status, Type, Pods, Namespace, and Cluster. The table contains the following data:

Name	Status	Type	Pods	Namespace	Cluster
mongodb-scope	OK	Deployment	1/1	default	scope-recette
nginx-ingress-controller	OK	Deployment	1/1	default	scope-recette
nginx-ingress-default-backend	OK	Deployment	1/1	default	scope-recette
scope-api	OK	Deployment	1/1	default	scope-recette
scope-backoffice	OK	Deployment	1/1	default	scope-recette
scope-front	OK	Deployment	1/1	default	scope-recette

FIGURE 4.23 – Création des pods

Les services

Dans notre cas, nous avons créé deux types de service, un type de service LoadBalancer qui permettra aux clients d'accéder à l'application de l'extérieur, et un service de type ClusterIP pour chaque pod qui assurera la communication entre ces derniers en interne. La figure 4.24 montre les services créés.

The screenshot shows the Google Cloud Services & Ingress interface. At the top, there are buttons for Refresh, Create Ingress, and Delete. Below that is a search bar and a toolbar with icons for Help. The main area displays a table of services. The columns are: Name, Status, Type, Endpoints, Pods, Namespace, and Clusters. The table contains the following data:

Name	Status	Type	Endpoints	Pods	Namespace	Clusters
mongodb-scope	OK	Cluster IP	...	1/1	default	scope-recette
nginx-ingress-controller	OK	External load balancer	...	1/1	default	scope-recette
nginx-ingress-default-backend	OK	Cluster IP	...	1/1	default	scope-recette
scope-api	OK	Cluster IP	...	1/1	default	scope-recette
scope-backoffice	OK	Cluster IP	...	1/1	default	scope-recette
scope-front	OK	Cluster IP	...	1/1	default	scope-recette

FIGURE 4.24 – Les services créés

Ingress

De même, nous avons créé un objet « Nginx LoadBalancer » de type Ingress qui assure le routage des Trafic http (externe) vers nos microservices exécutés dans le cluster ‘scope-recette’. La figure 4.25 présente les ingress de chaque micro-service.

The screenshot shows the Google Cloud Services & Ingress interface. The top navigation bar includes 'Google Cloud', a scope dropdown, a search bar ('Search Products, resources, docs (/)'), and a help icon. Below the header, there are dropdown menus for 'Cluster' and 'Namespace', and buttons for 'REFRESH', 'DELETE', 'RESET', and 'SAVE'. The main area is titled 'SERVICES' with a sub-tab 'INGRESS' selected. A descriptive text states: 'An Ingress is a collection of rules that allow inbound connections to reach the cluster services.' Below this is a filter section labeled 'Filter ingresses' with columns: Name (sorted by up arrow), Status, Type, Frontends, Services, Namespace, and Clusters. Three entries are listed:

Name	Status	Type	Frontends	Services	Namespace	Clusters
scope-api	OK	Custom	dev.scope.oyez.fr/api	scope-api	default	scope-recette
scope-backoffice	OK	Custom	admin.dev.scope.oyez.fr	scope-backoffice	default	scope-recette
scope-front	OK	Custom	dev.scope.oyez.fr	scope-front	default	scope-recette

FIGURE 4.25 – les ingress de chaque micro-service.

Storage

Dans notre projet, nous avons utilisé deux objets ‘storage’(volume) de type VPC (Persistent Volume Claims) – illustrer dans la figure 4.26, un pour stocker les données d’authentification à l’interface du micro-service ‘scope_backoffice’, tandis que l’autre est alloué au stockage des données entrantes (formulaire présentée dans l’interface ‘scope_backoffice’).

The screenshot shows the Google Cloud Storage interface. The top navigation bar includes 'Google Cloud', a scope dropdown, a search bar ('Search Products, resources, docs (/)'), and a help icon. Below the header, there are dropdown menus for 'Cluster' and 'Namespace', and buttons for 'REFRESH', 'DELETE', 'RESET', and 'SAVE'. The main area is titled 'Storage' with a sub-tab 'PERSISTENT VOLUME CLAIMS' selected. A descriptive text states: 'Persistent volume claims are requests for storage of specific size and access mode.' Below this is a filter section labeled 'Filter persistent volume claims' with columns: Name (sorted by up arrow), Phase, Volume, Storage class, Namespace, and Cluster. Two entries are listed:

Name	Phase	Volume	Storage class	Namespace	Cluster
mongo-scope-mongodb	Bound	pvc-46e31fcc-87b6-40a1-a29e-087a679fc2c5	standard	default	scope-recette
mongodb-scope	Bound	pvc-c2aedf1c-75ab-489f-8214-6b72a01c4845	standard	default	scope-recette

FIGURE 4.26 – Les ingress de chaque microservice.

4.4.4 Présentation des interfaces

La figure 4.27 montre l’interface du micro-service ‘scope_front’

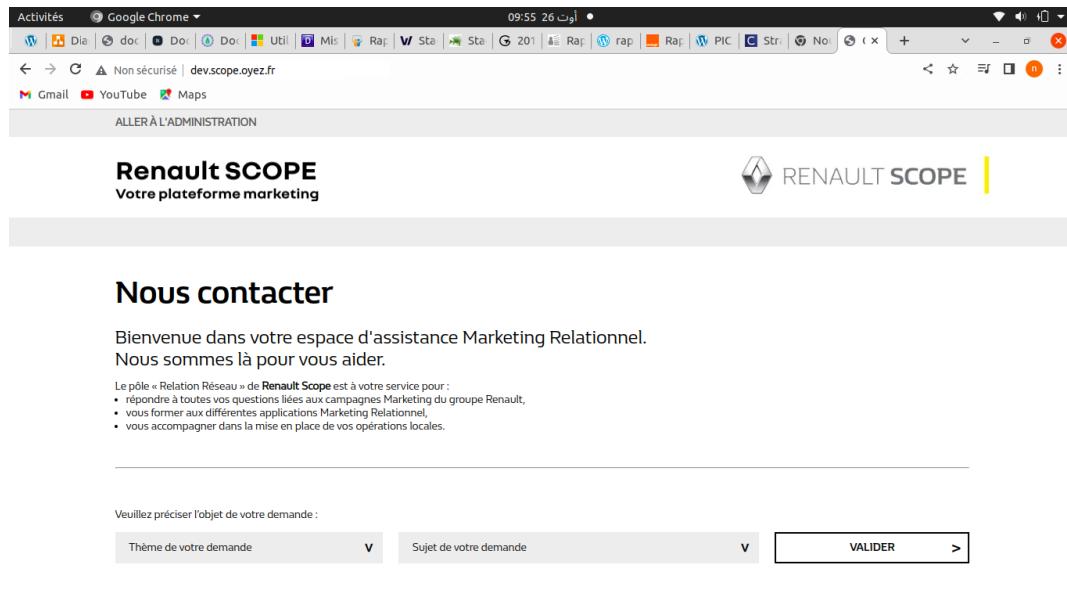


FIGURE 4.27 – Interface du micro-service ‘scope_front’

La figure 4.28 montre l’interface de login ‘scope_backoffice’

 A screenshot of a web browser showing the Renault SCOPE login page. The address bar shows 'http://admin.dev.scope.oyez.fr/admin/auth/login'. The main content area has a header 'RENault SCOPE' and a language switcher 'FR'. Below it, there are input fields for 'Prénom' (John), 'Nom' (Doe), 'Email' (@ kai@doe.com), and 'Mot de Passe'. There is also a field for 'Confirmation du mot de passe'. At the bottom, there is a checkbox for accepting terms and conditions and a blue 'PRÉT À COMMENCER' button.

FIGURE 4.28 – Interface de login ‘scope_backoffice’

Et la figure 4.29 présente l’interface graphique du micro-service ‘scope_backoffice’

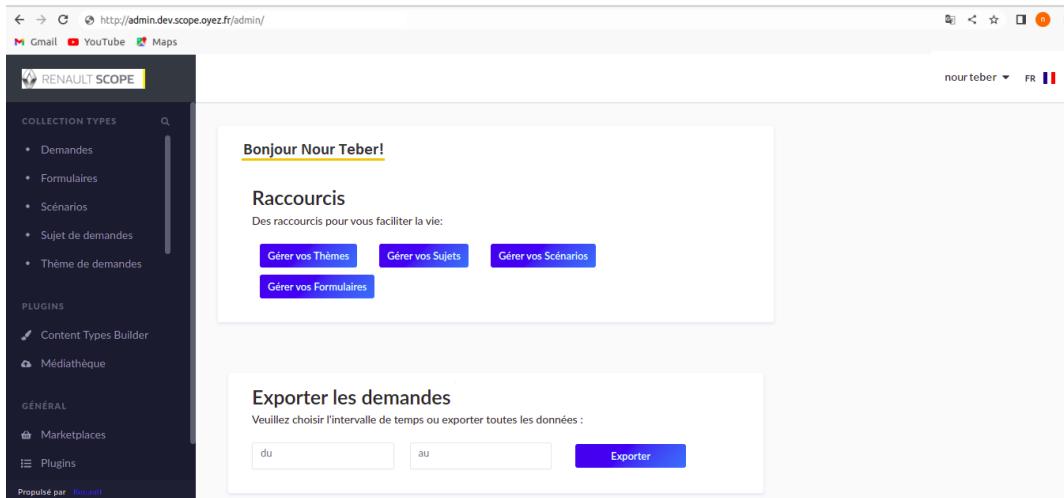


FIGURE 4.29 – Interface graphique 'scope_backoffice'

4.5 Conclusion

Durant ce dernier chapitre nous avons présenté l'architecture utilisée au déploiement des micro-services, ainsi que les outils utilisés et nous avons conclu par la réalisation pratique des étapes d'exécution.

Conclusion Générale

Notre projet de fin d'études était plus qu'une simple étape académique, c'était aussi l'occasion de travailler sur un projet d'envergure lié au DevOps. Ainsi, l'objectif principale de ce dernier est d'améliorer la capacité et le rendement des entreprises par rapport à l'utilisation des autres processus de développement. Ce stage a été réalisé au sein de l'entreprise Oyez-T sous l'encadrement de M. GHERIBI Mohamed Amine, d'où le projet s'intéresse à deux volets spécifiques, la conteneurisation de notre application et son déploiement avec kubernetes.

Ce rapport a été divisé en quatre chapitres bien définis :

- ❖ Nous avons consacré le premier chapitre à présenter l'organisation d'accueil, critiquant l'architecture actuelle adoptée au déploiement des produits en présentant notre solution proposée et même le démarche à suivre pour atteindre notre objectif.
- ❖ Ainsi que, le deuxième chapitre a été alloué pour définir le concept du DevOps, en dégageant ses avantages et ses pratiques, puis nous avons présenté l'architecture générale de notre solution et les outils nécessaires pour préambule son exécution dans les chapitres suivants.
- ❖ Ensuite, dans un 3ème chapitre nous avons présenté la première contribution de notre projet « La conteneurisation » dont son objectif est de conteneuriser les micro-services de notre application dans un environnement de DEV à l'aide de docker-compose, d'où nous avons illustré l'architecture proposée et les outils adoptés en finissant avec la démonstration des tâches exécuter à l'aides des captures.
- ❖ De même nous avons suivi la démarche du chapitre précédent pour présenter ce quatrième chapitre intitulé « », d'où nous avons présenté l'architecture proposée, les outils adoptés vois les captures des taché exécutés. L'objectif de ce chapitre est d'automatiser le déploiement de notre application sur un cluster GKE (Google Kubernetes Engine) à l'aide du pipeline CI/CD.

Sur le plan professionnel, ce projet nous a été très instructif du point de vue des connaissances acquises. Il a été très enrichissant puisqu'il nous a donné l'occasion de connaitre et utiliser une variété de concepts (DevOps, intégration continue, livraison continue...) et d'outils (Docker, Bitbucket, Git, Kubernetes, GCP...), et c'est ce qui nous motive à en savoir plus dans ce domaine. Finalement, nous souhaiterons que notre travail soit à la hauteur de vos attentes.

Webographie

- [1] <https://oyez.fr/>, Date : Juillet 2022
- [2] <https://www.guru99.com/agile-vs-devops.html>, Date : Juillet 2022
- [3] <https://www.redhat.com/fr/topics/devops/what-is-ci-cdquelle-est-la-diffrence-entre-ci-et-cd>, Date : Juillet 2022
- [4] <https://www.ionos.fr/digitalguide/sites-internet/developpement-web/git-ou-svn-comparaison-doutils-de-versionning/> :text=Les outils de versionning ont,moment charges voire restaures, Date : Juillet 2022
- [5] <https://www.journaldunet.fr/web-tech/guide-de-l-entreprise-digitale/1443816-bitbucket-l-outil-git-de-gestion-de-projet-et-de-livraison-continue/>, Date : Juillet 2022
- [6] <https://www.cloudflare.com/fr-fr/learning/cloud/what-is-the-cloud>, Date : Juillet 2022
- [8] <https://mobiskill.fr/blog/conseils-emploi-tech/aws-vs-gcp-quel-cloud-choisir-en-2022/>, Date : Juillet 2022
- [9] <https://geekflare.com/fr/docker-architecture>, Date : Aout 2022
- [10] https://kubernetes.io/fr/docs/concepts/_print/pg-45bdca6129cf540121623e903c18ba46, Date : Juillet 2022
- [11] <https://kubernetes.io/docs/concepts/workloads/pods>, Date : Aout 2022
- [12] <https://docs.microsoft.com/fr-fr/dotnet/architecture/microservices/container-docker-introduction/docker-containers-images-registries>, Date : Aout 2022
- [13] <https://net-security.fr/security/presentation-installation-dharbor-une-registry-securisee-pour-vos-conteneurs>, Date : Aout 2022
- [14] <https://cloud.google.com/vpc/docs/vpc?hl=fr> :text=Un réseau cloud privé virtuel,producton de Google Cloud avec Andromeda, Date : Aout 2022
- [15] https://cloud.google.com/compute/docs/access/iam?hl=frwhat_is_iam/, Date : Aout 2022

- [16] <https://cloud.google.com/container-registry?hl=fr/>, Date : Aout 2022
- [17] <https://cloud.netapp.com/blog/object-storage-block-and-shared-file-storage-in-google-cloud>, Date : Aout 2022
- [18] <https://www.cloudskillsboost.google/focuses/878?locale=frparent=catalog/>, Date : Aout 2022
- [19] <https://www.netapp.com/fr/devops-solutions/what-is-devops/>, Date : Aout 2022
- [20] <https://www.redhat.com/fr/topics/virtualization/>, Date : Aout 2022
- [21] <https://www.netapp.com/fr/devops-solutions/what-is-devops/>, Date : Aout 2022
- [22] <https://www.veritas.com/fr/ch/information-center/containerization>, Date : Aout 2022
- [23] <https://cloud.google.com/compute/docs/load-balancing-and-autoscaling>, Date : Aout 2022

Résumé

Dans le cadre de mon cursus en mastère co-construit en Sécurité des Réseaux et Communications Embarqués (SERCE) à l’Institut Supérieur de Technologie de l’Information et Communication (ISTIC Bordj Cédria), nous devons effectuer un stage de six mois au seins de l’entreprise “Oyez-T” sous le domaine de DevOps. Celui-ci doit nous permettre de connaître un véritable environnement de travail et surtout d’acquérir de nouvelles compétences. Dont, le sujet principal de mon stage était l’étude et la mise en place d’un pipeline CI/CD afin d’automatiser le déploiement d’une application conteneurisée et orchestrée par le service SaaS Google Kubernetes Engine (GKE) fournie par le fournisseur cloud Google Cloud Platform (GCP).

Mots clés : Kubernetes, GKE, Pipeline, CI/CD, GCP

Abstract

As part of my co-constructed master's studies in Network Security and Embedded Communications (SERCE) at the Higher Institute of Information Technologies and communication (ISTIC Bordj Cédria), I was required to do a six-month internship at the Institute at "Oyez-T" in the field of DevOps. This allows us to familiarize ourselves with the real work environment and above all to acquire new skills. The main subject of my internship was the study and implementation of a pipeline CI/CD to automate the deployment of a containerized and orchestrated application by the Google Kubernetes Engine (GKE) SaaS service provided by the cloud provider Google Cloud Platform (GCP).

Keywords : Kubernetes, GKE, Pipeline, CI/CD, GCP