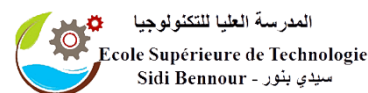




Université Chouaib Doukkali
École Supérieure de Technologie
Sidi Bennour



RAPPORT DE STAGE

En vue de l'obtention de la licence professionnelle

Filière :

Cyber Sécurité et Technologie Cloud

**Intégration Security-As-Code dans les déploiements
Kubernetes : Une approche automatisée avec GitLab-CI et
ArgoCD**

Présenté par :

- **Ayoub MOURADI**

Sous la direction du :

- **PR. Badreddine CHARKAOUI**
- **Mr. Yasser RADOUANI**

Membres du jury :

- **PR. Badreddine CHARKAOUI**
- **PR. EL Mehdi EL AROUSSI**

Remerciements

Je tiens à exprimer ma profonde gratitude et mes sincères remerciements à toutes les personnes qui ont contribué à la réussite de mon stage au sein de l'entreprise OCP Maintenance Solutions. Leur soutien inestimable, leur expertise et leurs encouragements ont grandement enrichi mon expérience professionnelle.

Je tiens tout d'abord à remercier chaleureusement mon encadrant de stage, M. Yasser RADOUANI, et M. Badr Eddine CHARKAOUI pour sa guidance et ses précieux conseils tout au long de mon stage. Sa patience, son professionnalisme et sa disponibilité ont été d'une aide précieuse dans la réalisation de mes missions.

Je tiens également à exprimer ma reconnaissance envers M. Karim ELBAZ, le responsable du département digital. Sa confiance en mes capacités et son soutien constant ont été des moteurs essentiels dans la réussite de mon stage. Ses orientations et son expertise ont été des sources d'inspiration pour mon projet professionnel.

Mes remerciements vont également à toute l'équipe de digitale pour leur accueil chaleureux, leur collaboration et leur partage de connaissances. Chaque membre de l'équipe a contribué à ma formation en me permettant de participer activement aux projets. Leur esprit d'équipe et leur bienveillance ont créé un environnement de travail stimulant et convivial. Un merci tout particulier à M. Abdellah LAMBARAA, dont le soutien constant et les précieux conseils ont été déterminants pour mon apprentissage.

Je souhaite exprimer ma reconnaissance envers tous mes collègues de travail pour leur soutien et leurs conseils précieux. Leurs expériences partagées, leurs encouragements et leur volonté de m'aider ont grandement facilité mon intégration et ma progression au sein de l'entreprise.

Je tiens à remercier l'ensemble du personnel de l'entreprise OCP Maintenance Solutions pour son amabilité et sa coopération. Leur accueil chaleureux et leur ouverture d'esprit ont contribué à rendre mon stage agréable et enrichissant.

Enfin, je tiens à remercier ma famille et mes amis pour leur soutien indéfectible tout au long de mon parcours académique et professionnel. Leur encouragement constant et leur soutien moral ont été essentiels pour mener à bien ce stage.

Liste des figures

FIGURE 1: LOGO DE GROUP OCP	6
FIGURE 2 LES SITES DE GROUP OCP	6
FIGURE 3 ZONE DE TRAITEMENT SITE JSF	8
FIGURE 4: USINE DE PHOSPHORE JSF	9
FIGURE 5: PLACE DE TRAITEMENT DES EAUX JSF	9
FIGURE 6 LOGO DE OCP MAINTENANCE SOLUTION	10
FIGURE 7 OMS SITE DE SAFI	10
FIGURE 8 SERVICE DE DIGITALISATION	11
FIGURE 9 INSPECTION ONLINE.....	12
FIGURE 10 INSPECTION OFFLINE	12
FIGURE 11 ORGANISATION.....	13
FIGURE 12 DIAGRAMME DE L'ARCHITECTURE D'UNE APPLICATION WEB.....	18
FIGURE 13 EXEMPLE DE COMPOSANT PRINCIPAL REACT POUR LA GESTION ET L'AFFICHAGE DE VALEURS NUMERIQUES.....	19
FIGURE 14 EXEMPLE DE FEUILLE DE STYLE CSS POUR LE COMPOSANT PRINCIPAL REACT.....	20
FIGURE 15 EXEMPLE DE COMPOSANT REACT POUR UNE PAGE SECONDAIRE AVEC NAVIGATION	20
FIGURE 16 EXEMPLE DE SERVEUR EXPRESS POUR LA GESTION DES VALEURS NUMERIQUES AVEC POSTGRESQL.....	21
FIGURE 17 EXEMPLE DE CONFIGURATION DES VARIABLES D'ENVIRONNEMENT POUR POSTGRESQL	22
FIGURE 18 EXEMPLE DE DOCKERFILE POUR DEPLOYER UNE APPLICATION NODE.JS AVEC POSTGRESQL	23
FIGURE 19 EXEMPLE DE DOCKERFILE POUR DEPLOYER UNE APPLICATION NODE.JS EN MODE DEVELOPPEMENT	23
FIGURE 20 EXEMPLE DE DOCKERFILE POUR DEPLOYER UN SERVEUR NGINX AVEC CONFIGURATION PERSONNALISEE	24
FIGURE 21 EXEMPLE DE CONFIGURATION DOCKER-COMPOSE POUR LE DEPLOIEMENT D'UNE APPLICATION FULL-STACK	25
FIGURE 22 LES COMMANDE DE GIT POUR POUSSER L'APPLICATION SUR UN REPOSITORY GITLAB.....	26
FIGURE 23 L'APPLICATION SUR UN REPOSITORY GITLAB.	26
FIGURE 24 EXEMPLE DE CONFIGURATION DE PIPELINE CI/CD AVEC GITLAB CI/CD	28
FIGURE 25 UN REPOSITORY DE MANIFESTE SUR GITLAB.	29
FIGURE 26 EXEMPLE DE MANIFESTE KUBERNETES POUR UN DEPLOIEMENT DE CLIENT.....	29
FIGURE 27 EXEMPLE DE MANIFESTE KUBERNETES POUR UN DEPLOIEMENT DE SERVEUR.....	30
FIGURE 28 EXEMPLE DE MANIFESTE KUBERNETES POUR UN DEPLOIEMENT DE POSTGRESQL	31
FIGURE 29 KUBERNETES	32
FIGURE 30 DIAGRAMME DE L'ARCHITECTURE D'UNE APPLICATION WEB SUR KUBERNETES	33
FIGURE 31 SORTI DE CMD MINIKUBE START	33
FIGURE 32 NGINX INGRESS CONTROLLER.....	34
FIGURE 33 APPLICATION DEPLOIEMENT.....	35
FIGURE 34 LES SERVICES D'APPLICATION.....	35
FIGURE 35 PROMETHEUS INTERFACE	36
FIGURE 36 GRAFANA INTERFACE.....	37
FIGURE 37 LIER GRAFANA AVEC PROMETHEUS	38
FIGURE 38 DASHBOARD DE CLUSTER KUBERNETES	38
FIGURE 39 RESULTAT DE ANALYSE UN FICHIER .YAML.....	39
FIGURE 40 DETECTION LES SERVICES.....	40
FIGURE 41 LES VULNERABILITES DE CLUSTER	40
FIGURE 42 ANALYSE LA CONFIGURATION DE SECURITE D'U MASTER	41
FIGURE 43 ANALYSE LA CONFIGURATION DE SECURITE ETCD.....	41
FIGURE 44 ANALYSE LA CONFIGURATION DE SECURITE NODE	41
FIGURE 44 ANALYSE LA CONFIGURATION DE SECURITE NODE	41

Sommaire

Introduction	5
Partie-1 : Organisme d'accueil	6
1-Présentation de l'organisme d'accueil	6
1-1-Le Groupe OCP	6
1-2-OCP Maintenance Solutions	10
Partie 2 : Projet réalisé	14
2-1-Pésentation de projet	14
2-1-1-Introduction :	14
2-1-2-Principes Fondamentaux de Security-as-Code :	14
2-2-Problématique	15
Contexte général :	15
Contexte spécifique :	15
Problématique :	15
Analyse des défis spécifiques :	16
Conclusion :	16
2-3-les technologie utilisée	17
2-4 Les parties du projet	18
Partie 1 : Création de l'application	18
Partie 2 : Conteneurisation de l'Application	23
Partie 3 : Gestion du Code Source	26
Partie 4 : La mise en place d'un pipeline GitLab CI	27
Partie 5 : Déploiement de l'Application	29
Partie 6 : Mise en Place de l'Environnement de Déploiement	32
Conclusion	42
Annexe	43
Bibliographie	44

Introduction

Dans le paysage technologique actuel, où les cybermenaces sont de plus en plus sophistiquées et fréquentes, l'intégration de la sécurité dès les premières phases du développement logiciel est devenue une pratique essentielle. Avec l'adoption généralisée des conteneurs et des systèmes d'orchestration tels que Kubernetes, il est impératif d'incorporer la sécurité de manière continue et automatisée pour garantir la robustesse et la résilience des applications tout au long de leur cycle de vie.

Kubernetes, en tant que plateforme d'orchestration de conteneurs, offre une flexibilité et une évolutivité exceptionnelles, mais cette complexité accrue présente également des défis uniques en matière de sécurité. Les déploiements Kubernetes doivent être soigneusement configurés pour éviter les vulnérabilités et assurer que les meilleures pratiques de sécurité soient suivies de manière cohérente.

Dans ce contexte, l'approche DevSecOps, qui intègre des pratiques de sécurité directement dans les pipelines de développement (CI/CD), émerge comme une solution efficace. En particulier, l'utilisation de la sécurité en tant que code (Security as Code) permet de traiter les configurations de sécurité et les politiques comme du code, ce qui facilite leur gestion et leur automatisation.

Ce rapport explore une approche novatrice pour intégrer la sécurité dans les déploiements Kubernetes en utilisant GitLab CI et ArgoCD. GitLab CI, une plateforme d'intégration continue et de livraison continue, permet d'automatiser les tests et les déploiements de sécurité. ArgoCD, une solution de déploiement continu pour Kubernetes, offre une gestion déclarative des applications, permettant une synchronisation continue entre le code et l'état de l'infrastructure déployée.

En intégrant ces outils, nous pouvons automatiser le processus de sécurité, détecter les vulnérabilités dès les premières phases du développement et assurer une sécurité proactive tout au long du cycle de vie de l'application. Cela permet non seulement de réduire les risques potentiels avant la mise en production, mais aussi d'offrir une visibilité accrue sur les configurations de sécurité et de faciliter les audits de conformité.

Ce rapport détaillera comment GitLab CI et ArgoCD peuvent être configurés pour renforcer la sécurité des environnements de déploiement, en assurant une intégration transparente et continue des contrôles de sécurité dans les workflows de développement. Nous examinerons également les bénéfices de cette approche, notamment en termes de réduction des vulnérabilités, de gestion des configurations de sécurité, et de conformité aux normes de sécurité.

Partie-1 : Organisme d'accueil

1-Présentation de l'organisme d'accueil

1-1-Le Groupe OCP



Figure 1: Logo de Group OCP

L'Office Chérifien des Phosphates (OCP) a été créé le 7 août 1920, sous forme d'un organisme d'État, mais étant donné le caractère de ses activités commerciales et industrielles, le législateur a tenu à le doter, dès sa création, d'une organisation lui permettant d'agir avec le même dynamisme et la même souplesse que les entreprises privées internationales, dans le monde.

Par la suite, l'évolution des activités de l'Office et l'ampleur de ses projets de valorisation ont conduit à la mise en place en 1974-1975, d'une structure de Groupe permettant l'intégration de différentes entités filiales complémentaires au sein d'un même ensemble : Groupe OCP.

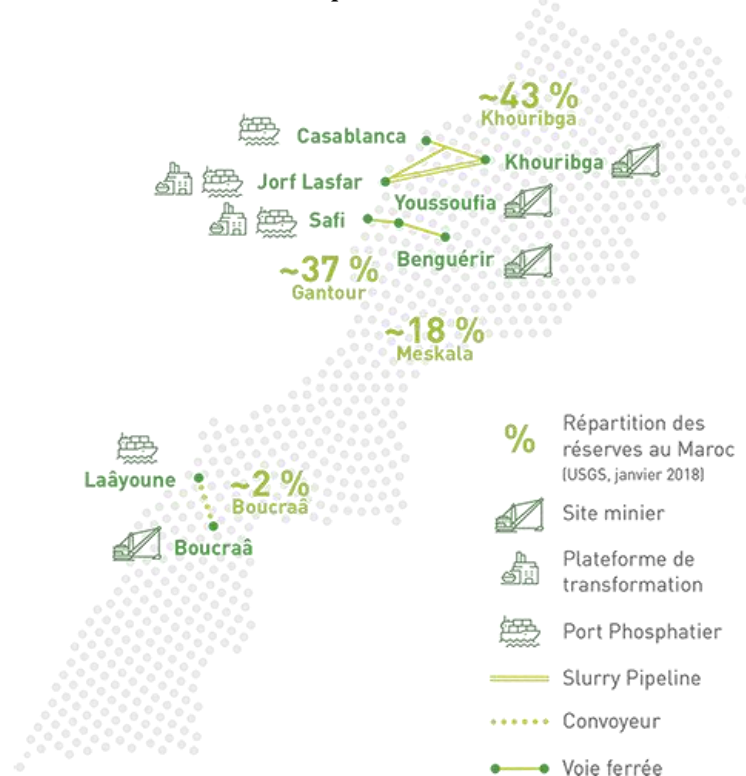


Figure 2 les sites de group ocp

1-1-2-Historique du groupe OCP

- **1920** : Création de l'Office Chérifien des Phosphate(OCP).
- **1921** : Début de l'extraction souterraine du phosphate dans les mines de Khouribga et première exportation du phosphate à partir du port de Casablanca.
- **1930** : Ouverture d'un nouveau centre de production de phosphate: le centre de Youssoufia, connu alors sous le nom de Louis Gentil.
- **1931** : Début de l'extraction souterraine du phosphate dans les mines de Youssoufia. **1951** : Démarrage de l'extraction à ciel ouvert à Sidi-Daoui (Khouribga) et début du développement des installations de séchage et de calcination à Khouribga.
- **1954** : Démarrage des premières installations de séchage à Youssoufia. **1958** : Création d'un centre de formation professionnelle à Khouribga. **1961** : Mise en service de la première laverie à Khouribga.
- **1965** : Lancement de la plateforme de transformation chimique de Safi (Maroc Chimie). **1974** : Lancement des travaux pour la réalisation du centre minier de Benguérir et naissance de l'Institut de Promotion Socio-Éducative (IPSE).
- **1975** : Intégration d'un nouveau centre minier appelé Phosboucraâ, création du Groupe OCP, intégration des industries chimiques aux structures internes du groupe OCP et création du Centre d'Études et de Recherches des Phosphates Minéraux (CERPHOS).
- **1976** : Démarrage à Safi de Maroc Phosphore I et Maroc Chimie.
- **1979** : Transfert des bureaux de la direction générale à Casablanca. **1980** : Ouverture de la mine de Benguérir.
- **1981** : Démarrage à Jorf Lasfar de Maroc Phosphore II.
- **1982** : Début des travaux de construction des complexes chimiques Maroc Phosphore III-IV à Jorf Lasfar.
- **1997** : Création de la société Indo-Maroc Phosphore (IMACID) en joint-venture avec le Groupe Birla.
- **1999** : Démarrage de la production d'acide phosphorique de l'usine d'IMACID à Jorf Lasfar.
- **2002** : Obtention par la mine de Benguérir du prix d'excellence JIPM (Japan Institute of Plant Maintenance).
- **2004** : Création de la société « Pakistan Maroc Phosphore SA » en joint-venture entre groupe OCP et la société Fauji Fertilizer Bin Qasim Limited du Pakistan.
- **2005** : Démarrage de l'usine de lavage/flottation de Youssoufia.
- **2008** : Transformation de l'Office Chérifien des Phosphates en société anonyme SA.
- **2009** : Augmentation du capital d'OCP SA d'un montant de 5 milliards de dirhams entièrement réservé à la Banque Centrale Populaire (BCP).

- **2010** : Création d'une joint-venture avec Jacobs engineering (JESA), ouverture d'un bureau de représentation au Brésil, ouverture d'un bureau de représentation en Argentine, création d'un fonds d'investissement agricole et création d'une société de valorisation du patrimoine immobilier (SADV ou Société d'Aménagement et de Développement vert).
- **2011** : Lancement des travaux de l'usine de dessalement à Jorf Lasfar.
- **2013** : Lancement de la construction de quatre nouvelles usines de granulation, ayant chacune une capacité de production annuelle d'un million de tonnes à Jorf Lasfar et création de Dupont OCP Operations Consulting S.A. (Dupont OCP).
- **2014** : Lancement du projet Slurry Pipeline entre Khouribga et les centres industriels et ouverture d'un bureau de représentation à Singapour.
- **2015** : Ouverture d'un bureau de représentation en Côte d'Ivoire et lancement d'OCP Africa.
- **2018** : Lancement de la marque PHOSFEED et d'une nouvelle branche de stockage au Brésil.

Le site de Safi

La direction des industries chimiques de SAFI : filiale du groupe OCP, a pour vocation la transformation d'une partie des phosphates de Youssoufia et les phosphates en provenance de Ben Guérir pour la fabrication de l'acide phosphorique et des engrais. Ce complexe, qui emploie 3448 agents, permet de consommer jusqu'à 6 millions de tonnes de phosphate. Une partie de cette production est transformée localement en engrais TSP et en phosphates alimentaires pour bétail et volaille (MCP-DCP), l'autre partie est exportée sous forme de différentes qualités d'acide phosphorique à une multitude de Clients. Les flux des produits et matières premières sont acheminés par voie ferroviaire entre le site et la mine d'une part, le site et le port de Safi d'autre part.

1-1-3-MAROC CHIMIE

Maroc Chimie a servi comme foyer de formation dans lequel l'encadrement d'une partie du personnel a été nécessaire aux autres usines. Puisée, elle se compose principalement d'atelier sulfurique, atelier phosphorique, atelier d'engrais.



Figure 3 Zone de traitement site JSF

1-1-4-MAROC PHOSPHORE I

L'usine qui a démarré en 1976, sur la base de l'utilisation du phosphate de Youssoufia et du soufre importé pour la production d'acide phosphorique elle se compose essentiellement des six ateliers : Atelier sulfurique, Atelier phosphorique, central et utilités, Atelier MAP, Atelier fusion Filtre, Action de soufre.



Figure 4: usine de phosphore JSF

1-1-5-PHOSPHORE II

Cette division a démarré en 1981. Elle comporte quatre ateliers : Atelier sulfurique, Atelier énergie et fluide, Atelier phosphorique, Atelier laverie

1-1-6-Le Groupe OCP au cœur de la transformation digitale

Pour le groupe OCP, être compétitif dans un environnement concurrentiel vif et un marché en perpétuelle évolution est un enjeu majeur. Grâce au digital, le Groupe peut répondre à ces défis et orienter sa stratégie de leadership entièrement tournée vers une agriculture durable. Le digital permettra au groupe OCP de saisir les opportunités, de conserver son agilité commerciale en proposant de nouveaux produits innovants et d'accroître sa flexibilité industrielle créatrice de valeur pour faire face à toutes formes de concurrence. À cette fin, L'OCP se dote d'une feuille de route concernant la digitalisation. Après la transformation industrielle, la transformation digitale est en marche.



Figure 5: place de traitement des eaux JSF

1-2-OCP Maintenance Solutions



Figure 6 logo de OCP maintenance solution

Projet phare de la maintenance digitalisée né de plusieurs initiatives innovantes au sein du Groupe, OCP Maintenance Solutions piloté par une équipe projet au sein du site de Safi a conduit à la création d'une Business Unit en 2017 au service du consulting en management des équipements et en développement logiciel au sein du Groupe OCP, spécialisée dans la fiabilité et dans les solutions de maintenance 4.0 avancées pour plusieurs secteurs de service. Grâce à l'utilisation de technologies de pointe en IOT, Big Data, Analyse et Machine Learning, elle a développé des outils logiciels très fiables pour la maintenance préventive 4.0 adaptés aux besoins de ses clients. OCP Maintenance Solutions offre une variété de services et de produits à grande valeur ajoutée pour prédire et aider ses clients industriels internes et externes à réaliser des plans de maintenance optimisés et améliorer la disponibilité de la production en réduisant considérablement le risque des arrêts non planifiés. La BusinessUnit renforce sa présence nationale en créant des bureaux dans tout le Maroc par l'intermédiaire de l'ensemble des sites industriels appartenant au Groupe OCP.



Figure 7 OMS site de Safi

1-2-1-Les services d'OCP Maintenance solutions

A. Accompagnement

Riche d'une expérience mesurée en décennies, L'OCP a acquis une expertise confirmée en maintenance. Sachant qu'il dispose d'une large sélection d'équipements tournants allant du plus basique : convoyeur, motopompe. Au plus sophistiqué : turbine, broyeur, granulateur... OCP MS a recueilli cette expertise et l'a rehaussé en certifiant les meilleurs éléments OCP en les ralliant à OCP MS. OCP Maintenance solutions dispose en effet d'experts certifiés en soudure, vibration, pomperie, robinetterie ..., qui sont les principaux domaines de toute industrie. Qui sont prêts pour accompagner les clients et les aider à optimiser leurs opérations de maintenance.

B. Digitalisation

La digitalisation est l'impact sur les entreprises et les organisations du fait que les gens et les objets sont interconnectés en permanence, en tout lieu et pour tous les usages. Le concept est fondé sur des bases des années 2000 et 2010 avec l'arrivée d'internet puis la numérisation des données. La digitalisation fait suite à cela, avec une mise à niveau des entreprises avec l'utilisation permanente des nouvelles technologies, des méthodes et usages du web. Ces changements touchèrent les moyens mis en place pour atteindre les objectifs de performance. Le besoin de digitaliser les services de production s'impose comme une évidence. Aussi la digitalisation de la fonction maintenance, au même titre que celle de la production ou de la qualité, tient un rôle fondamental dans cette quête de la performance globale, offrant des gains tangibles en matière de sécurité, de productivité et de rentabilité. La digitalisation contribue également au décloisonnement des services de l'entreprise en rendant l'information accessible à l'ensemble des acteurs concernés, tout en facilitant la mobilité des équipes.



Figure 8 service de digitalisation

C. Formations industrielles

La formation continue est essentielle pour les agents de maintenance afin d'accomplir leurs tâches quotidiennes et s'épanouir dans leur travail. Cependant ils doivent surement être fatigués des formations traditionnelles fastidieuses qui les noient par des informations peu pratiques, des informations qu'ils n'utiliseront probablement pas dans leurs tâches quotidiennes. Gestion de données

Inspections des équipements tournants

De nos jours être à l'écoute des variations d'états de la machine est un impératif, un suivi rigoureux du Park machine s'impose. C'est pour cela qu'OCP MS propose deux types de services :

Un service online :

Ce service avec l'installation d'un nombre de capteurs sans fil permettant le relevé de vibration, accélération, spectre, et température, à une fréquence très élevée, ces données permettent d'anticiper les pannes et de les planifier en bonne et due forme. C'est une entrée aussi dans le monde du digital dans la mesure où ces données permettront de créer des modèles analytiques et donc de prédire rigoureusement la date prévue de la panne.



Figure 9 Inspection online

Un service offline :

Ce service consiste en des rondes d'inspections effectuées par les experts MS. La collecte de mesures de vibration et de température se fait manuellement, mais est consolidée au niveau 6 d'une plateforme informatique. Les clients pourront donc consulter le détail de la ronde et l'état de leur parc machine sur cette plateforme.

Techniques CND (Contrôle non destructif)



Figure 10 inspection offline

L'objectif premier des CND est de répondre aux impératifs de sécurité, gestion des risques et qualité. En maintenance les CND sont utilisés pour évaluer la qualité de l'intervention de maintenance, ou détecter d'éventuels défauts non visibles à l'œil nu ou inaccessibles. Ces diagnostics permettent de décider rigoureusement sur les actions à entreprendre. Souvent en industrie, un mauvais diagnostic engendrerait des pertes énormes. Il est donc impératif de contacter des experts en CND pour garantir la fiabilité des diagnostics proposés

1-2-2-Organigramme

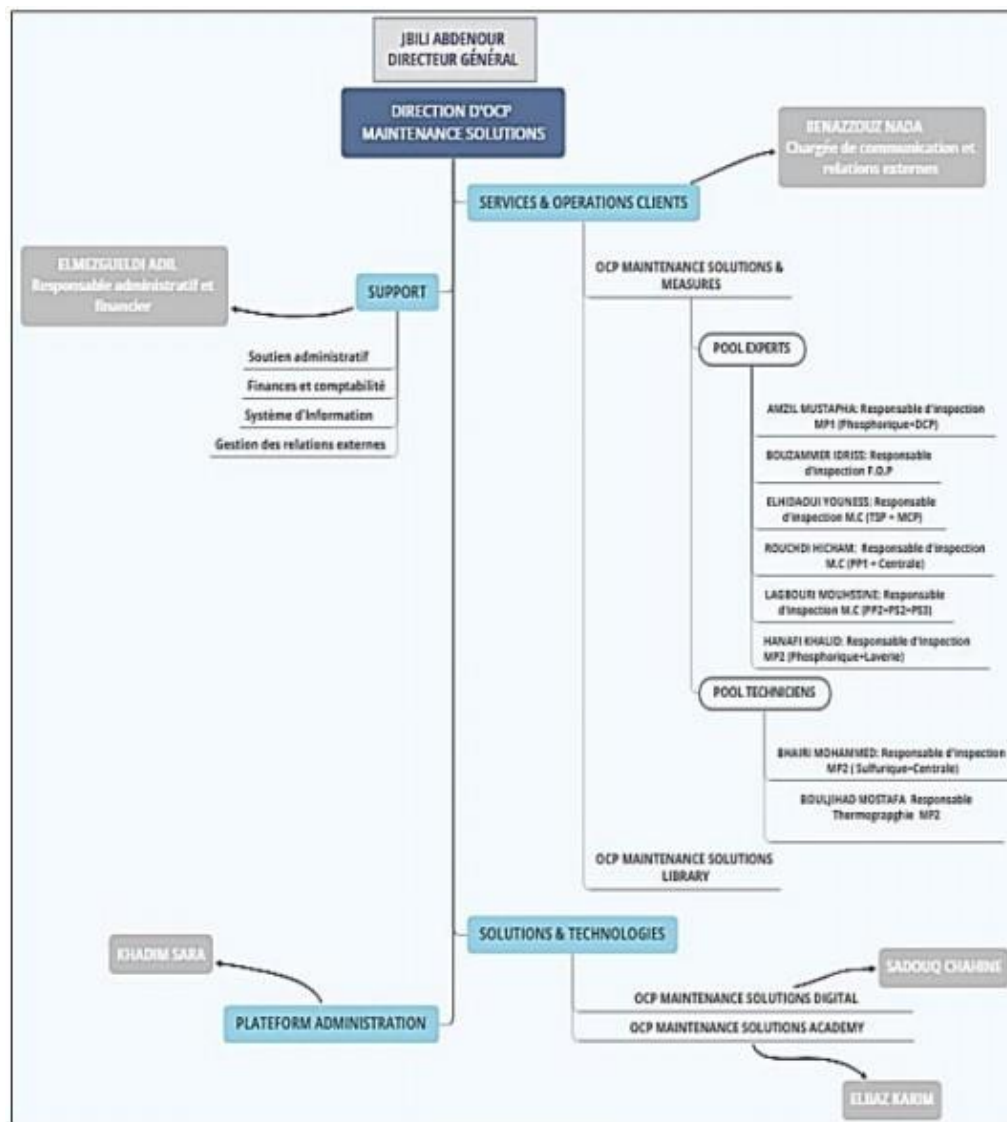


Figure 11 organisation

Site web:
<https://www.ocp-ms.com>

Secteur:
Services et conseil aux entreprises

Partie 2 : Projet réalisé

2-1-Pésentation de projet

2-1-1-Introduction :

"Security-as-Code" est un concept qui se réfère à l'approche consistant à intégrer les principes de sécurité dès la conception et tout au long du développement d'un logiciel ou d'un système informatique. Cela implique d'adopter des pratiques de programmation sécurisées, telles que la validation des entrées, la gestion des erreurs et l'identification des vulnérabilités potentielles dès les premières étapes du processus de développement. En adoptant cette approche, on vise à minimiser les risques liés à la sécurité informatique et à garantir la protection des données et des utilisateurs.

2-1-2-Principes Fondamentaux de Security-as-Code :

a) Intégration Continue de la Sécurité:

L'approche SaC vise à intégrer la sécurité dans chaque phase du développement logiciel, depuis la conception jusqu'à la mise en production. Les équipes de développement et de sécurité collaborent étroitement pour identifier et corriger les vulnérabilités dès les premières étapes.

b) Automatisation des Tests de Sécurité:

L'automatisation est un pilier central du SaC. Les tests de sécurité automatisés permettent de vérifier en continu la sécurité du code, de détecter les vulnérabilités potentielles et de s'assurer que les modifications apportées au code n'introduisent pas de nouvelles failles.

c) Gestion des Erreurs et Validation des Entrées

Les pratiques de programmation sécurisées incluent la validation des entrées pour empêcher les attaques par injection et d'autres types de vulnérabilités. La gestion des erreurs doit être rigoureuse pour éviter les divulgations d'informations sensibles.

b) Surveillance et Audits de Sécurité

La mise en place de mécanismes de surveillance continue et d'audits réguliers est essentielle pour détecter et répondre rapidement aux incidents de sécurité. Les outils d'audit automatisés peuvent analyser le code et les configurations pour identifier les risques.

2-2-Problématique

Sécurisation et Automatisation des Déploiements Kubernetes avec une Approche Security-As- Code

Contexte général :

Avec l'adoption massive de Kubernetes pour l'orchestration des conteneurs et la gestion des microservices, les entreprises peuvent désormais déployer des applications de manière plus flexible et évolutive. Cependant, cette flexibilité s'accompagne de nouveaux défis en matière de sécurité et de gestion des configurations. Les environnements Kubernetes sont souvent complexes et dynamiques, ce qui peut rendre difficile la détection et la gestion des vulnérabilités.

Dans ce contexte, il est crucial d'intégrer la sécurité dès les premières étapes du développement et de l'intégrer de manière continue dans le cycle de déploiement. C'est là que la notion de Security-As-Code prend tout son sens : il s'agit d'automatiser les pratiques de sécurité dans les pipelines CI/CD pour garantir que les applications et les infrastructures sont sécurisées à chaque étape du développement et du déploiement.

Contexte spécifique :

Pour tester cette approche, une application Node.js avec une base de données PostgreSQL a été développée. Voici les étapes réalisées :

- Création de l'application et de ses configurations Docker (Dockerfile et docker-compose). Pousser l'image de l'application sur Docker Hub.
- Pousser le code de l'application sur GitLab avec un fichier .gitlab-ci.yml incluant les outils de sécurité suivants : SonarCloud, Trivy, Snyk, OWASP.
- Création d'un dépôt de manifests Kubernetes pour l'application.
- Installation et configuration de ArgoCD, Minikube, Helm, et Nginx Ingress Controller.
- Utilisation de Prometheus et Grafana pour la surveillance.
- Installation de kube-bench, kube-hunter, et Checkov pour les évaluations de sécurité.

Problématique :

Comment garantir une sécurité optimale et continue des déploiements Kubernetes en intégrant des outils de sécurité dans un pipeline CI/CD automatisé, tout en assurant une visibilité complète et une réponse proactive aux vulnérabilités détectées ?

Analyse des défis spécifiques :

1. Automatisation des vérifications de sécurité :

- Défi : Intégrer divers outils de sécurité dans le pipeline CI/CD pour détecter les vulnérabilités le plus tôt possible.
- Solution : Utilisation de GitLab-CI pour automatiser les scans de sécurité avec SonarCloud (analyse de code), Trivy (analyse des images Docker), Snyk (analyse des dépendances), et OWASP (vérification des failles de sécurité web).

2. Gestion des configurations et des secrets :

- Défi : Sécuriser les configurations et les secrets sensibles utilisés par les applications déployées sur Kubernetes.
- Solution : Utilisation de Kubernetes Secrets pour gérer les informations sensibles et Helm pour les configurations standardisées et sécurisées des déploiements.

3. Surveillance et réponse aux incidents :

- Défi : Mettre en place une surveillance continue pour détecter et répondre rapidement aux incidents de sécurité.
- Solution : Intégration de Prometheus et Grafana pour la surveillance des performances et des incidents, ainsi que kube-hunter pour la détection des vulnérabilités au niveau du cluster.

4. Visibilité et reporting :

- Défi : Assurer une visibilité complète sur l'état de sécurité de l'application et de l'infrastructure Kubernetes.
- Solution : Utilisation de rapports générés par les outils de sécurité intégrés dans le pipeline CI/CD et les tableaux de bord Grafana pour la visualisation en temps réel.

5. Formation et sensibilisation :

- Défi : Sensibiliser les équipes de développement et d'opérations aux pratiques de sécurité DevSecOps.
- Solution : Formation continue et documentation sur les outils et les pratiques de sécurité utilisés, ainsi que des revues régulières des processus de sécurité.

Conclusion :

En intégrant ces outils de sécurité dans le pipeline CI/CD avec GitLab-CI et ArgoCD, nous avons pu automatiser et renforcer la sécurité des déploiements Kubernetes. Cette approche Security-As-Code permet de détecter et de corriger les vulnérabilités de manière proactive, assurant ainsi une meilleure protection des applications et de l'infrastructure contre les menaces de sécurité. La surveillance continue et la gestion centralisée des configurations et des secrets contribuent également à maintenir un environnement sécurisé et conforme aux meilleures pratiques de sécurité.

2-3 les technologie utilisée

Application:

- React (front-end)
- Node.js (back-end)
- PostgreSQL (base de données)
- Nginx (serveur Web)

Pipeline CI/CD:

- GitLab CI
- SonarCloud (analyse de code statique)
- Trivy (analyse de vulnérabilités des images Docker)
- Snyk (analyse des dépendances)
- OWASP ZAP (analyse de sécurité Web)

Infrastructure:

- Kubernetes (orchestration de conteneurs)
- Argo CD (déploiement continu)
- Nginx Ingress Controller (routage des requêtes HTTP/HTTPS)
- Helm (gestionnaire de paquets Kubernetes)

Monitoring:

- Prometheus (surveillance des métriques)
- Grafana (visualisation des données de surveillance)

Sécurité:

- Checkov (analyse de configuration de l'infrastructure comme code)
- kube-hunter (test de sécurité des clusters Kubernetes)
- kube-bench (test de sécurité de la configuration Kubernetes)
- Starboard (gestion des vulnérabilités Kubernetes)

2-4 Les parties du projet

Partie 1 : Création de l'application

Dans le cadre de mon stage, j'ai développé et déployé une application web simple en utilisant une stack technologique moderne comprenant React pour le frontend, Node.js pour le backend, PostgreSQL comme base de données et Nginx comme serveur web. Ce projet avait pour objectif principal de tester les capacités de déploiement de l'application sur Kubernetes

Architecture de l'Application

L'application est structurée en trois couches principales :

Frontend (React) :

Développé en React, le frontend de l'application offre une interface utilisateur simple et réactive. Il permet aux utilisateurs d'interagir avec l'application et de soumettre des données.

Backend (Node.js) :

Le backend, construit avec Node.js et Express.js, gère les requêtes HTTP provenant du frontend, traite les données et interagit avec la base de données PostgreSQL.

Base de Données (PostgreSQL) :

PostgreSQL est utilisé comme système de gestion de base de données relationnelle pour stocker les données de l'application de manière structurée et sécurisée.

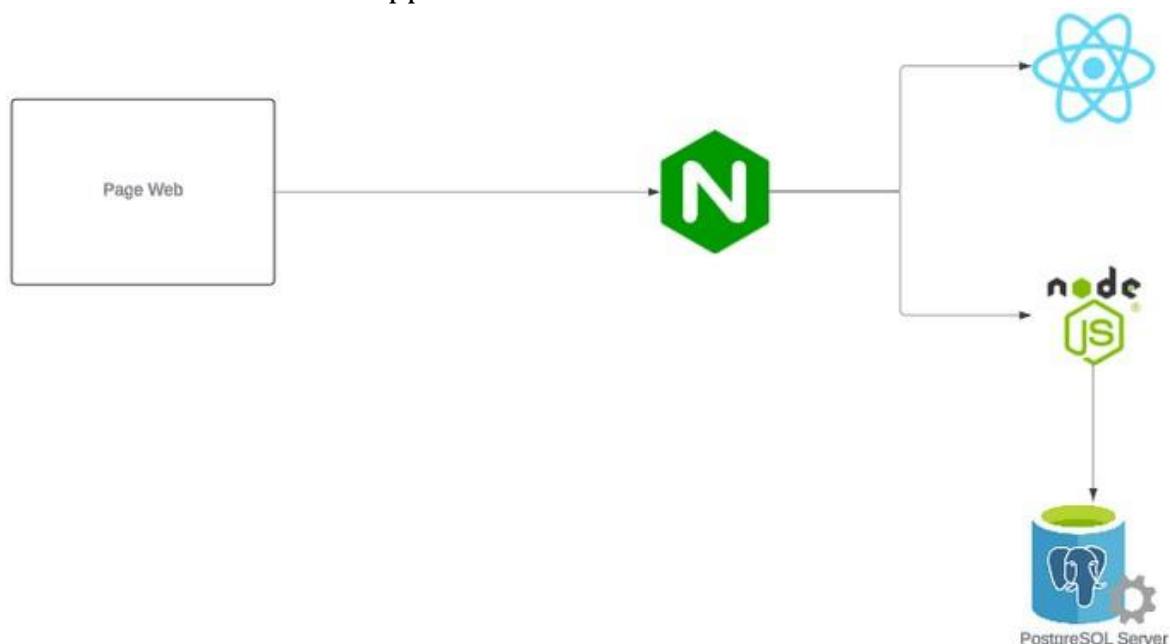


Figure 12 Diagramme de l'Architecture d'une Application Web

Frontend :

```
1 import { useCallback, useState, useEffect } from "react";
2 import axios from "axios";
3 import "./MainComponent.css";
4 const MainComponent = () => {
5   const [values, setValues] = useState([]);
6   const [value, setValue] = useState("");
7   const getAllNumbers = useCallback(async () => {
8     const data = await axios.get("/api/values/all");
9     setValues(data.data.rows.map(row => row.number));
10  }, []);
11  const saveNumber = useCallback(
12    async event => {
13      event.preventDefault();
14
15      await axios.post("/api/values", {
16        value
17      });
18
19      setValue("");
20      getAllNumbers();
21    },
22    [value, getAllNumbers]
23  );
24  useEffect(() => {
25    getAllNumbers();
26  }, []);
27  return (
28    <div>
29      <button onClick={getAllNumbers}>Get all numbers</button>
30      <br />
31      <span className="title">Values</span>
32      <div className="values">
33        {values.map(value => (
34          <div className="value">{value}</div>
35        ))}
36      </div>
37      <form className="form" onSubmit={saveNumber}>
38        <label>Enter your value: </label>
39        <input
40          value={value}
41          onChange={event => {
42            setValue(event.target.value);
43          }}
44        />
45        <button>Submit</button>
46      </form>
47    </div>
48  );
49 };
50 export default MainComponent;
```

Figure 13 Exemple de Composant Principal React pour la Gestion et l’Affichage de Valeurs Numériques

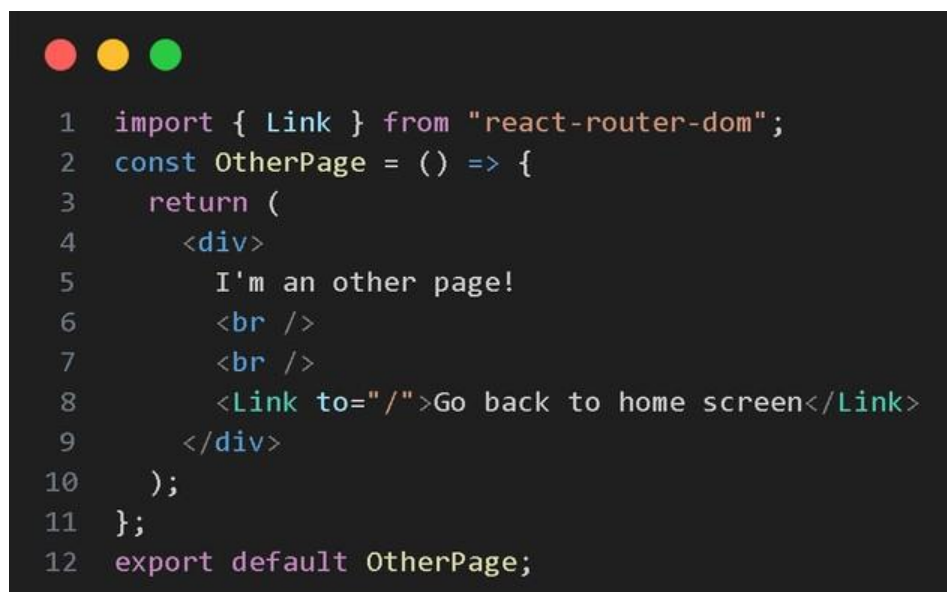
Ce CSS stylise les éléments du composant principal React, incluant les titres, les valeurs affichées, et le formulaire de soumission. Voici le code CSS utilisé :



```
1  .title {  
2    font-weight: bold;  
3  }  
4  .values {  
5    margin-top: 20px;  
6    background: yellow;  
7  }  
8  .value {  
9    margin-top: 10px;  
10   border-top: 1px dashed black;  
11 }  
12 .form {  
13   margin-top: 20px;  
14 }
```

Figure 14 Exemple de Feuille de Style CSS pour le Composant Principal React

Ce composant React représente une page secondaire avec une fonctionnalité de navigation permettant de retourner à la page d'accueil. Voici le code du composant :

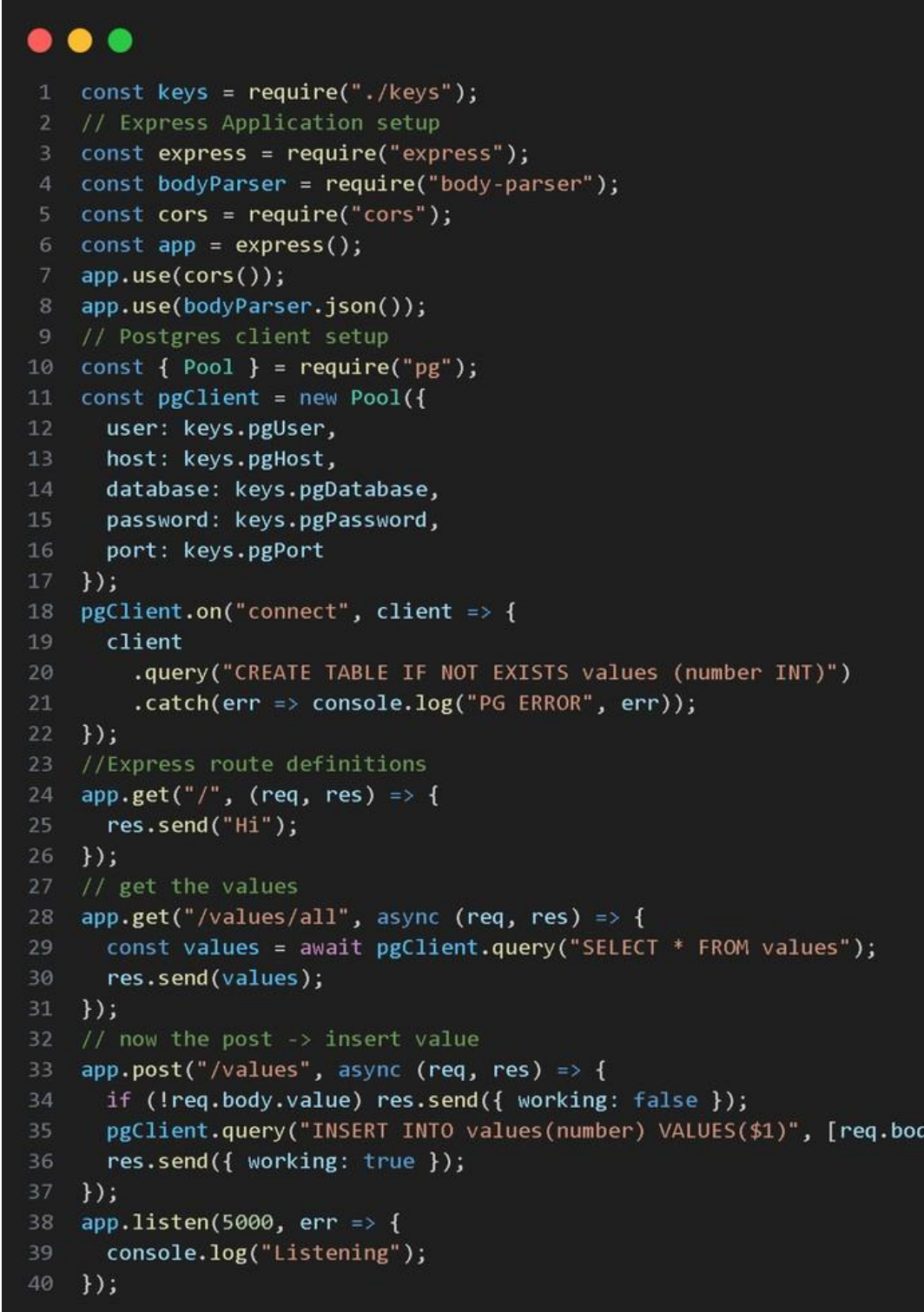


```
1  import { Link } from "react-router-dom";  
2  const OtherPage = () => {  
3    return (  
4      <div>  
5        I'm an other page!  
6        <br />  
7        <br />  
8        <Link to="/">Go back to home screen</Link>  
9      </div>  
10   );  
11 };  
12 export default OtherPage;
```

Figure 15 Exemple de Composant React pour une Page Secondaire avec Navigation

Backend :

Ce code met en place un serveur Express qui gère les valeurs numériques stockées dans une base de données PostgreSQL. Il utilise les bibliothèques express, body-parser et cors pour gérer les requêtes et les réponses, et pg pour interagir avec la base de données PostgreSQL.



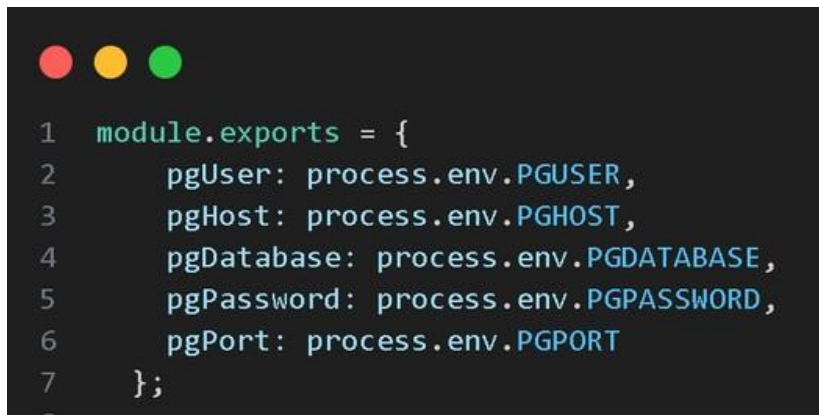
```
1  const keys = require("../keys");
2  // Express Application setup
3  const express = require("express");
4  const bodyParser = require("body-parser");
5  const cors = require("cors");
6  const app = express();
7  app.use(cors());
8  app.use(bodyParser.json());
9  // Postgres client setup
10 const { Pool } = require("pg");
11 const pgClient = new Pool({
12   user: keys.pgUser,
13   host: keys.pgHost,
14   database: keys.pgDatabase,
15   password: keys.pgPassword,
16   port: keys.pgPort
17 });
18 pgClient.on("connect", client => {
19   client
20     .query("CREATE TABLE IF NOT EXISTS values (number INT)")
21     .catch(err => console.log("PG ERROR", err));
22 });
23 //Express route definitions
24 app.get("/", (req, res) => {
25   res.send("Hi");
26 });
27 // get the values
28 app.get("/values/all", async (req, res) => {
29   const values = await pgClient.query("SELECT * FROM values");
30   res.send(values);
31 });
32 // now the post -> insert value
33 app.post("/values", async (req, res) => {
34   if (!req.body.value) res.send({ working: false });
35   pgClient.query("INSERT INTO values(number) VALUES($1)", [req.body.value]);
36   res.send({ working: true });
37 });
38 app.listen(5000, err => {
39   console.log("Listening");
40 });
```

Figure 16 Exemple de Serveur Express pour la Gestion des Valeurs Numériques avec PostgreSQL

Ce module configure les variables d'environnement nécessaires pour se connecter à une base de données PostgreSQL. Les variables sont extraites des variables d'environnement du système pour une configuration flexible et sécurisée.

Description:

Le fichier `keys.js` exporte un objet contenant les configurations PostgreSQL, en utilisant les variables d'environnement du système pour définir les paramètres de connexion. Cela permet de sécuriser les informations sensibles et de les configurer facilement selon l'environnement (développement, test, production).

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in JavaScript and defines an object for PostgreSQL configuration using process.env variables. The code is as follows:

```
1  module.exports = {  
2    pgUser: process.env.PGUSER,  
3    pgHost: process.env.PGHOST,  
4    pgDatabase: process.env.PGDATABASE,  
5    pgPassword: process.env.PGPASSWORD,  
6    pgPort: process.env.PGPORT  
7  };
```

Figure 17 Exemple de Configuration des Variables d'Environnement pour PostgreSQL

Partie 2 : Conteneurisation de l'Application

Pour garantir la portabilité et faciliter le déploiement de l'application :

- Créé des Dockerfiles pour chaque service de l'application (frontend, backend, base de données).
- Construit les images Docker de l'application.
- Poussé les images sur Docker Hub pour les rendre accessibles pour le déploiement.

Dockerfile pour le frontend

Dockerfile configure un environnement Docker pour déployer une application Node.js qui interagit avec une base de données PostgreSQL. Voici le code détaillé du Dockerfile :

```
FROM node:14.21.0-alpine
WORKDIR /app
COPY ./package.json ./
RUN npm i
COPY . .
CMD ["npm", "run", "start"]
```



Figure 18 Exemple de Dockerfile pour Déployer une Application Node.js avec PostgreSQL

Ce Dockerfile, combiné avec les autres composants décrits (comme le serveur Express et les configurations PostgreSQL), permet de créer une application Node.js robuste et facilement déployable.

Dockerfile pour le backend

Ce Dockerfile configure un environnement Docker pour déployer une application Node.js en mode développement. Voici le code détaillé du Dockerfile :

```
FROM node:14.21.0-alpine
WORKDIR /app
COPY ./package.json ./
RUN npm i
COPY . .
CMD ["npm", "run", "dev"]
```

Figure 19 Exemple de Dockerfile pour Déployer une Application Node.js en Mode Développement

Ce Dockerfile simplifie le processus de déploiement et de développement en automatisant la configuration de l'environnement Docker pour l'application Node.js. Il permet aux développeurs de travailler dans un environnement isolé et cohérent, facilitant ainsi le développement et le déploiement de l'application.

Dockerfile pour Nginx

Dockerfile configure un environnement Docker pour déployer un serveur NGINX avec une configuration personnalisée. Voici le code détaillé du Dockerfile :



```
1  #Nginx
2  FROM nginx
3  COPY ./default.conf /etc/nginx/conf.d/default.conf
```

Figure 20 Exemple de Dockerfile pour Déployer un Serveur NGINX avec Configuration Personnalisée

Ce Dockerfile simplifie le déploiement d'un serveur NGINX en automatisant la configuration de l'environnement Docker. Il permet également d'utiliser une configuration personnalisée pour adapter le comportement du serveur NGINX selon les besoins spécifiques de l'application.

Commande

Pour construire l'image Docker, utilisez la commande suivante dans le répertoire contenant le Dockerfile

Construction de l'Image Docker

```
docker build -f Dockerfile.dev -t mayoub12/multi-client
docker build -f Dockerfile.dev -t mayoub12/multi-server1
```

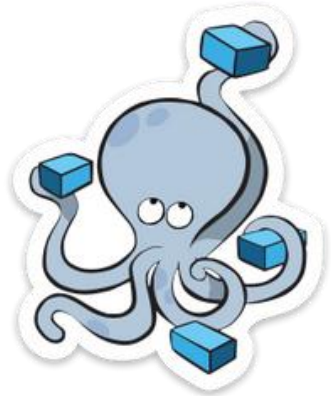
Exécution du Conteneur Docker

```
docker run -it -p 4003:5000 mayoub12/multi-server1
docker run -it -p 4002:3000 mayoub12/multi-client
```


Docker Compose

Docker Compose est un outil de gestion d'orchestration de conteneurs open-source qui simplifie le déploiement et la gestion d'applications multi-conteneurs Docker

```
1  #docker-compose.yml
2  version: "3"
3  services:
4    postgres:
5      image: "postgres:latest"
6      environment:
7        - POSTGRES_PASSWORD=postgres_password
8    nginx:
9      depends_on:
10       - api
11       - client
12      restart: always
13      build:
14        dockerfile: Dockerfile.dev
15        context: ./nginx
16      ports:
17        - "3050:80"
18    api:
19      build:
20        dockerfile: Dockerfile.dev
21        context: "./server"
22      volumes:
23        - /app/node_modules
24        - ./server:/app
25      environment:
26        - PGUSER=postgres
27        - PGHOST=postgres
28        - PGDATABASE=postgres
29        - PGPASSWORD=postgres_password
30        - PGPORT=5432
31    client:
32      stdin_open: true
33      environment:
34        - CHOKIDAR_USEPOLLING=true
35      build:
36        dockerfile: Dockerfile.dev
37        context: ./client
38      volumes:
39        - /app/node_modules
40        - ./client:/app
```



Commande

```
docker-compose --build
```

Figure 21 Exemple de Configuration docker-compose pour le déploiement d'une Application Full-Stack

Cette configuration docker-compose simplifie le déploiement d'une application full-stack en automatisant la création et l'exécution de plusieurs conteneurs Docker. Elle permet une gestion efficace des dépendances entre les services et facilite le partage du code source entre l'hôte et les conteneurs.

Partie 3 : Gestion du Code Source

J'utilise GitLab pour la gestion du code source. Les actions effectuées incluent :

- Pousser l'application sur un repository GitLab.
- Créer un fichier gitlab-ci.yml pour configurer les pipelines d'intégration continue (CI).

Git: est un système de contrôle de version distribué utilisé pour suivre les modifications dans les fichiers informatiques et coordonner le travail entre plusieurs personnes sur des projets logiciels. Il permet aux développeurs de travailler simultanément sur des versions de code, de fusionner leurs modifications et de gérer les versions de manière efficace.

GitLab: est une plateforme de gestion de projet basée sur Git. Il offre des fonctionnalités supplémentaires telles que la gestion des problèmes, le suivi du temps, l'intégration continue, le déploiement continu, etc. GitLab fournit également un hébergement pour les référentiels Git, permettant aux équipes de collaborer sur leurs projets et de gérer tout le processus de développement logiciel depuis une seule interface.



Figure 22 les commandes de git pour pousser l'application sur un repository GitLab.

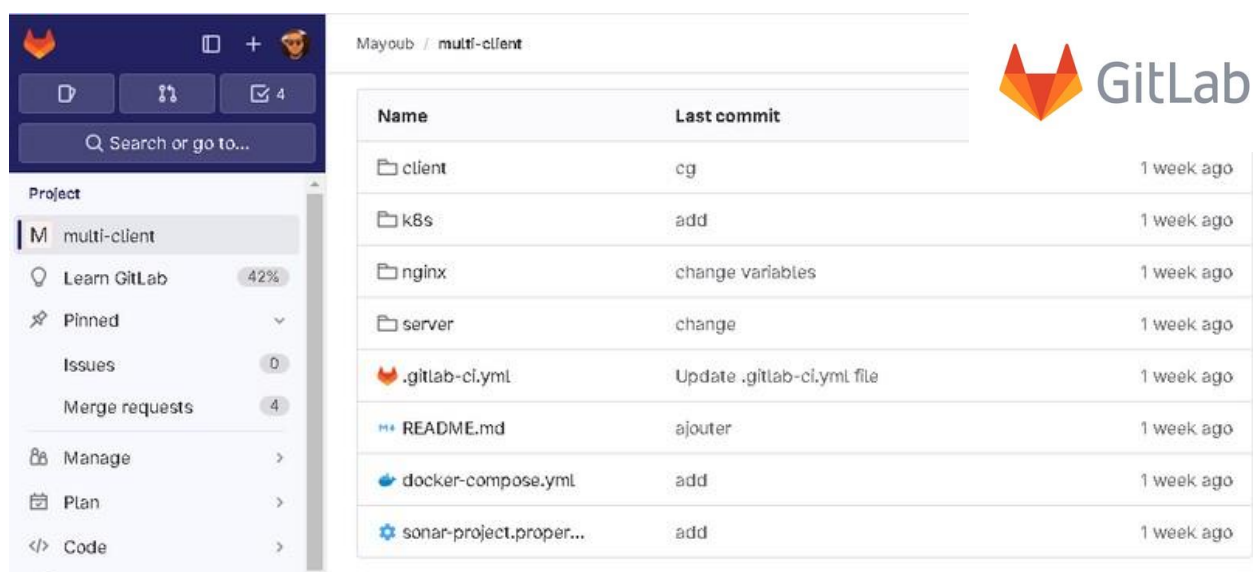


Figure 23 l'application sur un repository GitLab.

Partie 4 : La mise en place d'un pipeline GitLab CI

GitLab CI (Continuous Integration) est un service intégré de GitLab qui permet d'automatiser le processus d'intégration continue dans le développement logiciel. Il offre un ensemble d'outils et de fonctionnalités pour automatiser la construction, les tests et le déploiement des applications à chaque modification de code. Grâce à GitLab CI, les développeurs peuvent configurer des pipelines d'intégration continue qui comprennent des étapes telles que la compilation du code source, l'exécution de tests automatisés, l'analyse de sécurité et le déploiement automatique. Cela permet d'améliorer la qualité du code, d'accélérer le processus de développement et de garantir la fiabilité des applications tout au long de leur cycle de vie.



Les outils intégrés à mon pipeline incluent SonarCloud pour l'analyse statique du code, Trivy pour la détection des vulnérabilités dans les conteneurs, Snyk pour la gestion des dépendances, et OWASP ZAP pour les tests d'intrusion automatisés.

SonarCloud est une plateforme d'analyse statique du code qui fournit des informations détaillées sur la qualité du code, en identifiant les bugs, les vulnérabilités de sécurité, les mauvaises pratiques de codage et les odeurs de code. Il permet d'améliorer la qualité et la sécurité du code en fournissant des recommandations spécifiques pour résoudre les problèmes détectés.



Trivy est un scanner de vulnérabilités open source conçu pour les conteneurs et les images de conteneurs Docker. Il recherche les images Docker à la recherche de vulnérabilités connues dans les dépendances et les bibliothèques, ce qui permet de détecter et de corriger les problèmes de sécurité dès le début du processus de développement.

Snyk est une plateforme de gestion des vulnérabilités des dépendances qui aide à identifier, évaluer et résoudre les vulnérabilités dans les bibliothèques et les packages utilisés dans le code. Il offre des fonctionnalités de surveillance continue des dépendances pour s'assurer que les applications restent sécurisées tout au long de leur cycle de vie.



OWASP ZAP (Zed Attack Proxy) est un outil de test de sécurité automatisé utilisé pour identifier les vulnérabilités dans les applications web. Il peut détecter une gamme de problèmes de sécurité, y compris les injections SQL, les failles XSS, les vulnérabilités CSRF et bien d'autres, ce qui permet aux équipes de développement de corriger rapidement les problèmes avant qu'ils ne soient exploités par des attaquants.

. gitlab-ci.yml

Ce script définit les étapes et les tâches nécessaires pour automatiser le processus d'intégration continue et de déploiement continu (CI/CD) d'une application. Voici une explication détaillée du script



```
.gitlab-ci.yml 2.41 KB  Blame  Edit  Replace  Delete

1  stages:
2    - scan-fs-trivy
3    - sonarcloud
4    - build
5    - scan-image
6    - scan-dépendances
7
8  variables:
9    DOCKER_CLIENT: mayou012/multi-client-oms:latest
10   DOCKER_SERVER: mayou012/multi-server-oms:latest
11   SONAR_USER_NAME: "${CI_PROJECT_DIR}/.sonar" # Defines the location of the analysis task cache
12   GIT_DEPTH: "0" # Tells git to fetch all the branches of the project, required by the analysis task
13
14   #-----docker-images-----#
15   build_client:
16     stage: build
17     image: docker:latest
18     script:
19       - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
20       # - docker-compose up --build
21       - cd client
22       - docker build -t $DOCKER_CLIENT .
23       - docker push $DOCKER_CLIENT
24
25   build_server:
26     stage: build
27     image: docker:latest
28     script:
29       - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
30       - cd server
31       - docker build -t $DOCKER_SERVER .
32       - docker push $DOCKER_SERVER
33
34   #-----Scan-fs-trivy-----#
35   trivy-scann-fs:
36     stage: scan-fs-trivy
37     script:
38       - trivy fs --format table --output trivy_results_fs.csv
39     only:
40       - merge_requests
41       - main
42     artifacts:
43       paths:
44         - trivy_results_fs.csv
45
46   #-----SonarCloud-----#
47   sonarcloud-check:
48     stage: sonarcloud
49     image:
50       name: sonarsource/sonar-scanner-cli:latest
51       entrypoint: [""]
52     cache:
53       key: "${CI_JOB_NAME}"
54       paths:
55         - .sonar/cache
56     script:
57       - sonar-scanner
58     only:
59       - merge_requests
60       - master
61       - develop
62
63   #-----Scan-Image(Trivy)-----#
64   trivy-scann:
65     stage: scan-image
66     script:
67       - trivy image --format table --output trivy_results_client.csv $DOCKER_CLIENT
68       - trivy image --format table --output trivy_results_server.csv $DOCKER_SERVER
69     only:
70       - merge_requests
71       - main
72     artifacts:
73       paths:
74         - trivy_results_client.csv
75         - trivy_results_server.csv
76
77   #-----Snyk-----#
78   snyk-scan:
79     stage: scan-dépendances
80     script:
81       - snyk test --detailed > resultat_snyk.txt
82       - snyk test --severity=high > resultat_snyk_critiques.txt
83       # snyk protect #pro-package-lock.json
84     only:
85       - merge_requests
86       - main
87     artifacts:
88       paths:
89         - resultat_snyk.txt
90         - resultat_snyk_critiques.txt
91
92
93
```

Figure 24 Exemple de Configuration de Pipeline CI/CD avec GitLab CI/CD

Partie 5 : Déploiement de l'Application

Pour le déploiement de l'application, j'ai :

- Créé un repository de manifestes Kubernetes.
- Rédigé des fichiers de déploiement et de services en format YAML.
- Poussé ces fichiers sur GitLab pour le déploiement.

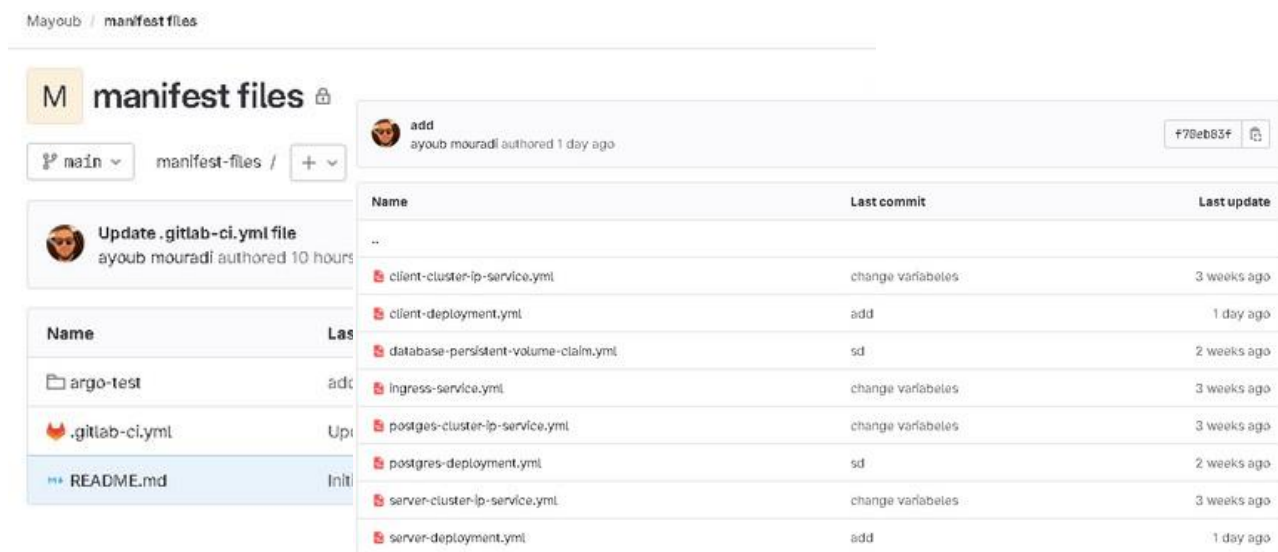


Figure 25 un repository de manifeste sur GitLab.

Manifeste Kubernetes

Client

Manifeste Kubernetes définit un déploiement pour l'application client

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client-deployment
spec:
  replicas: {{.Values.replicaCount}}
  selector:
    matchLabels:
      component: web
  template:
    metadata:
      labels:
        component: web
    spec:
      containers:
        - name: client
          image: mayoub12/multi-client
          ports:
            - containerPort: 3000
```

Figure 26 Exemple de Manifeste Kubernetes pour un Déploiement de Client

Ce manifeste Kubernetes peut être utilisé pour déployer et gérer l'application client dans un cluster Kubernetes. Il permet de définir les spécifications de déploiement, y compris le nombre de répliques et les images à utiliser.

Serveur

manifeste Kubernetes définit un déploiement pour l'application serveur.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: server-deployment
5  spec:
6    replicas: {{ .Values.replicaCount }}
7    selector:
8      matchLabels:
9        component: server
10   template:
11     metadata:
12       labels:
13         component: server
14     spec:
15       containers:
16         - name: server
17           image: mayoub12/multi-server1
18           ports:
19             - containerPort: 5000
20           env:
21             - name: PGUSER
22               value: postgres
23             - name: PGHOST
24               value: postgres-cluster-ip-service
25             - name: PGPORT
26               value: "5432"
27             - name: PGDATABASE
28               value: postgres
29             - name: PGPASSWORD
30               valueFrom:
31                 secretKeyRef:
32                   name: pgpassword
33                   key: PGPASSWORD
34
```

Figure 27 Exemple de Manifeste Kubernetes pour un Déploiement de Serveur

Ce manifeste Kubernetes peut être utilisé pour déployer et gérer l'application serveur dans un cluster Kubernetes. Il permet de définir les spécifications de déploiement, y compris le nombre de répliques, les images à utiliser et les variables d'environnement nécessaires pour la connexion à la base de données PostgreSQL.

PostgreSQL

Ce manifeste Kubernetes définit un déploiement pour le serveur PostgreSQL.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: postgres-deployment
5  spec:
6    replicas: {{ .Values.replicaCount }}
7    selector:
8      matchLabels:
9        component: postgres
10   template:
11     metadata:
12       labels:
13         component: postgres
14     spec:
15       volumes:
16         - name: postgres-storage
17           persistentVolumeClaim:
18             claimName: database-persistent-volume-claim
19       containers:
20         - name: postgres
21           image: postgres
22           ports:
23             - containerPort: 5432
24           volumeMounts:
25             - name: postgres-storage
26               mountPath: /var/lib/postgresql/data
27               subPath: postgres
28           env:
29             - name: POSTGRES_PASSWORD
30               valueFrom:
31                 secretKeyRef:
32                   name: pgpassword
33                   key: PGPASSWORD
```

Figure 28 Exemple de Manifeste Kubernetes pour un Déploiement de PostgreSQL

Ce manifeste Kubernetes peut être utilisé pour déployer et gérer un serveur PostgreSQL dans un cluster Kubernetes. Il permet de définir les spécifications de déploiement, y compris le nombre de répliques, les images à utiliser, les volumes persistants et les variables d'environnement nécessaires pour configurer le serveur PostgreSQL.

Partie 6 : Mise en Place de l'Environnement de Déploiement

Introduction

Pour la mise en place de l'environnement de déploiement, j'ai orchestré la configuration et l'installation de plusieurs outils essentiels. Cela comprend ArgoCD, un outil de gestion des déploiements continus qui simplifie et automatise le processus de déploiement des applications. J'ai également mis en place Minikube, un outil permettant de créer un cluster Kubernetes local, offrant ainsi un environnement de développement et de test complet. Pour faciliter la gestion des ressources Kubernetes, j'ai déployé Helm, un gestionnaire de packages Kubernetes, permettant de créer, partager et gérer des applications basées sur Kubernetes via des charts. Enfin, j'ai configuré Nginx Ingress Controller, une solution pour gérer les règles d'ingress dans le cluster Kubernetes, assurant ainsi un routage efficace du trafic vers les applications déployées.

De plus, j'ai intégré Prometheus et Grafana pour la surveillance et la visualisation des métriques du cluster Kubernetes, offrant une visibilité accrue sur la performance et l'état des applications. Pour renforcer la sécurité et la conformité, j'ai déployé Checkov, un outil d'analyse statique de la sécurité des infrastructures en tant que code, ainsi que Kube-hunter, un outil de test de pénétration de sécurité spécifique à Kubernetes. J'ai également installé Kube-bench pour effectuer des audits de sécurité basés sur les recommandations du Centre pour la Sécurité Internet (CIS) Kubernetes Benchmark. Enfin, j'ai utilisé Starboard pour centraliser et visualiser les résultats des différents outils de sécurité au sein du cluster Kubernetes.

Kubernetes

Kubernetes, souvent abrégé en K8s, est une plateforme open source de gestion d'orchestration de conteneurs. Il permet de déployer, de mettre à l'échelle et de gérer des applications conteneurisées de manière efficace et automatisée. Kubernetes fournit un environnement robuste pour le déploiement et la gestion d'applications dans des conteneurs, en offrant des fonctionnalités telles que l'auto-scaling, la répartition de charge, la gestion des ressources, le déploiement déclaratif, la surveillance des applications et la gestion des mises à jour. Grâce à son architecture modulaire et à sa flexibilité, Kubernetes est devenu un outil essentiel dans le domaine du développement logiciel moderne, permettant aux équipes de développeurs de déployer et de gérer des applications de manière efficace, fiable et évolutive.



Figure 29kubernetes

A. Infrastructure

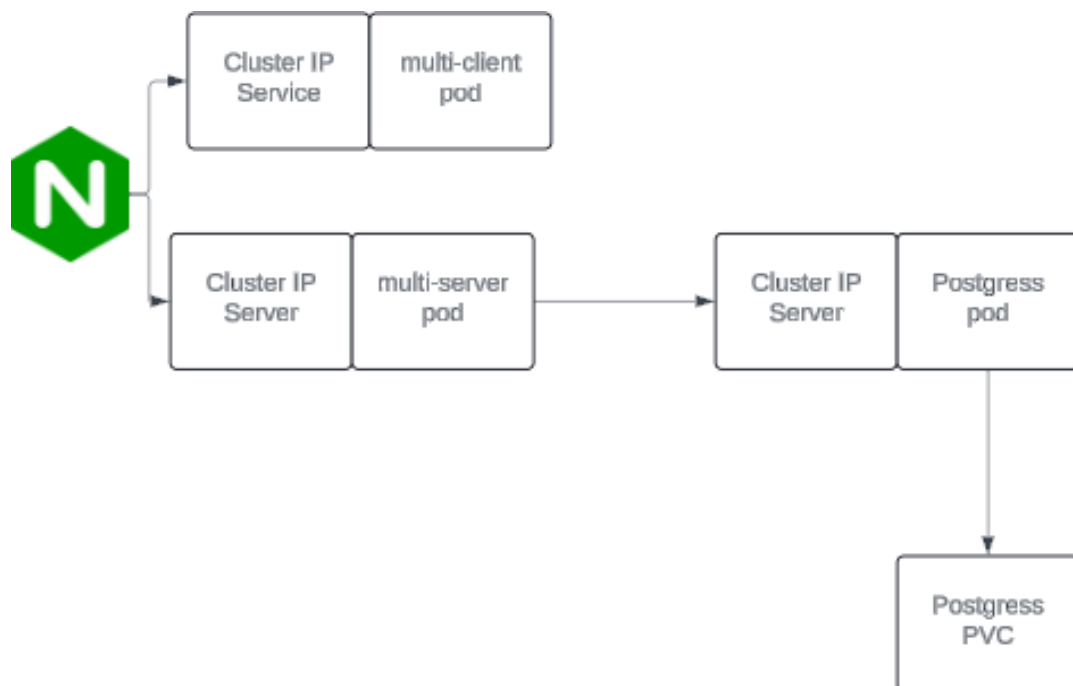


Figure 30 Diagramme de l'Architecture d'une Application Web sur Kubernetes

Minikube :

Minikube est un outil open source qui permet de créer un cluster Kubernetes local sur une machine virtuelle. Il est souvent utilisé pour le développement et le test d'applications Kubernetes en fournissant un environnement isolé et facile à configurer.



Pour installer Minikube, vous pouvez utiliser les commandes suivantes :

```
$curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
```

```
$sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

Pour démarrer Minikube, utilisez :

```
$minikube start
```

Pour plus d'informations, vous pouvez consulter la documentation officielle sur <https://minikube.sigs.k8s.io/docs/>.

```
C:\Users\ayoub> minikube start
9527 20:55:35.137478 14532 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:
Users\ayoub\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: The system cannot find the path speci
fied.
minikube v1.33.0 sur Microsoft Windows 10 Pro 10.0.19045.4412 Build 19045.4412
minikube 1.33.1 est disponible ! Téléchargez-le ici : https://github.com/kubernetes/minikube/releases/tag/v1.33.1
Pour désactiver cette notification, exécutez : 'minikube config set WantUpdateNotification false'

Utilisation du pilote docker basé sur le profil existant
Démarrage du nœud "minikube" primary control-plane dans le cluster "minikube"
Extraction de l'image de base v0.0.43...
> gcr.io/k8s-minikube/kicbase...: 35.11 MiB / 480.29 MiB 7.31% 2.54 MiB p
```

Figure 31 Sortie de la commande minikube start

Helm :

Helm est un gestionnaire de packages pour Kubernetes qui simplifie le déploiement et la gestion d'applications Kubernetes. Il permet d'emballer les applications Kubernetes dans des charts, qui sont des packages contenant tous les fichiers nécessaires au déploiement d'une application.



Pour installer Helm, vous pouvez utiliser la commande suivante :

```
$sudo apt-get install helm
```

```
$helm version
```

Pour plus d'informations, vous pouvez consulter la documentation officielle sur <https://helm.sh/docs/>.

```
PS C:\Users\ayoub> helm version
version.BuildInfo{Version:"v3.14.4", GitCommit:"81c902a123462fd4052bc5e9aa9c513c4c8fc142", GitTreeState:"clean", GoVersion:"go1.21.9"}
```

Nginx Ingress Controller :

Nginx Ingress Controller est un contrôleur Ingress pour Kubernetes qui gère les règles d'acheminement du trafic entrant vers les services Kubernetes. Il fonctionne en tant que reverse proxy et permet de configurer des règles d'acheminement basées sur des noms de domaine, des



chemins, etc., facilitant ainsi la mise en place de l'exposition des services Kubernetes à l'extérieur du cluster.

Pour installer le Nginx Ingress Controller, vous pouvez utiliser les commandes suivantes :

```
$kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml
```

Pour plus d'informations, vous pouvez consulter la documentation officielle sur <https://kubernetes.github.io/ingress-nginx/>.

```
PS C:\Users\ayoub> kubectl get pods -n ingress-nginx
NAME                                READY   STATUS    RESTARTS   AGE
ingress-nginx-admission-create-kqdsn 0/1     Completed 0           34h
ingress-nginx-admission-patch-lc5hc   0/1     Completed 1           34h
ingress-nginx-controller-57b7568757-qqlh5 1/1     Running   0           34h
```

Figure 32nginx ingress controller

ArgoCD :

ArgoCD est un outil open source utilisé pour la livraison continue et la gestion des déploiements sur des clusters Kubernetes. Il automatise le déploiement des applications et assure le suivi de l'état souhaité défini dans les fichiers de configuration.



Pour installer ArgoCD, vous pouvez utiliser les commandes suivantes :

\$kubectl create namespace argocd

\$kubectl apply -n argocd -f <https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml>

Pour accéder à l'interface utilisateur Web d'ArgoCD, exécutez :

\$kubectl port-forward svc/argocd-server -n argocd 8080:443

Pour plus d'informations, vous pouvez consulter la documentation officielle sur <https://argo-cd.readthedocs.io/>.

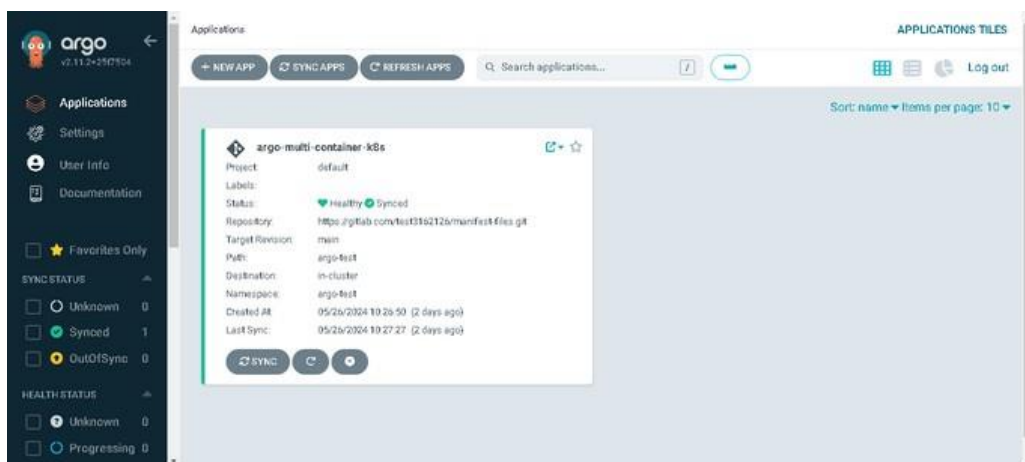


Figure 33 Application déploiement

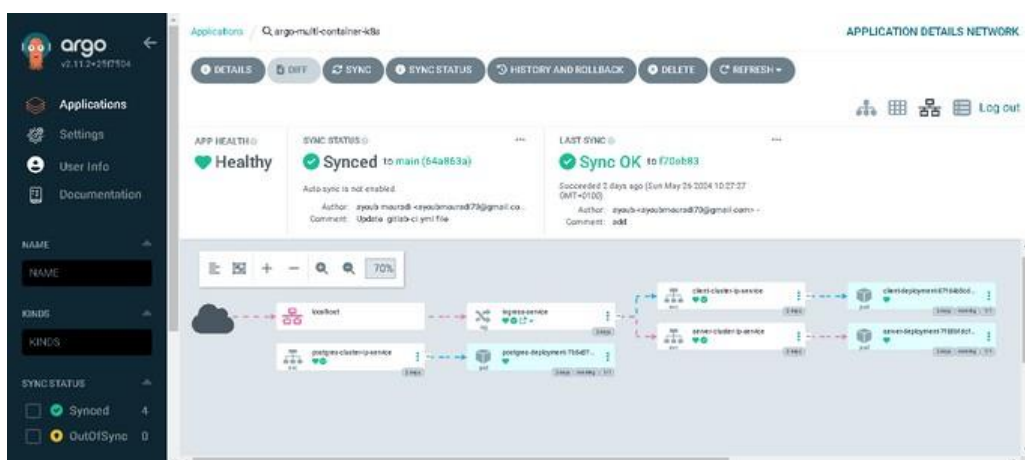


Figure 34 Les services d'application

B. Monitoring

Prometheus

Prometheus est un système open-source de surveillance et d'alerte conçu pour collecter et stocker des métriques de séries chronologiques. Il est largement utilisé dans les environnements Kubernetes pour surveiller les performances des applications et des clusters.



Pour plus d'informations, vous pouvez consulter la documentation officielle sur <https://prometheus.io/docs/>.

Pour installer Prometheus vous pouvez utiliser les commandes suivantes :(helm)

```
$helm search hub Prometheus
```

```
$helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

```
$helm repo update
```

```
$helm install prometheus prometheus-community/prometheus
```

```
$kubectl get service
```

```
$kubectl expose service prometheus-server --type=NodePort --target-port=9090 --name=prometheus-server-ext
```

```
$minikube service prometheus-server-ext
```

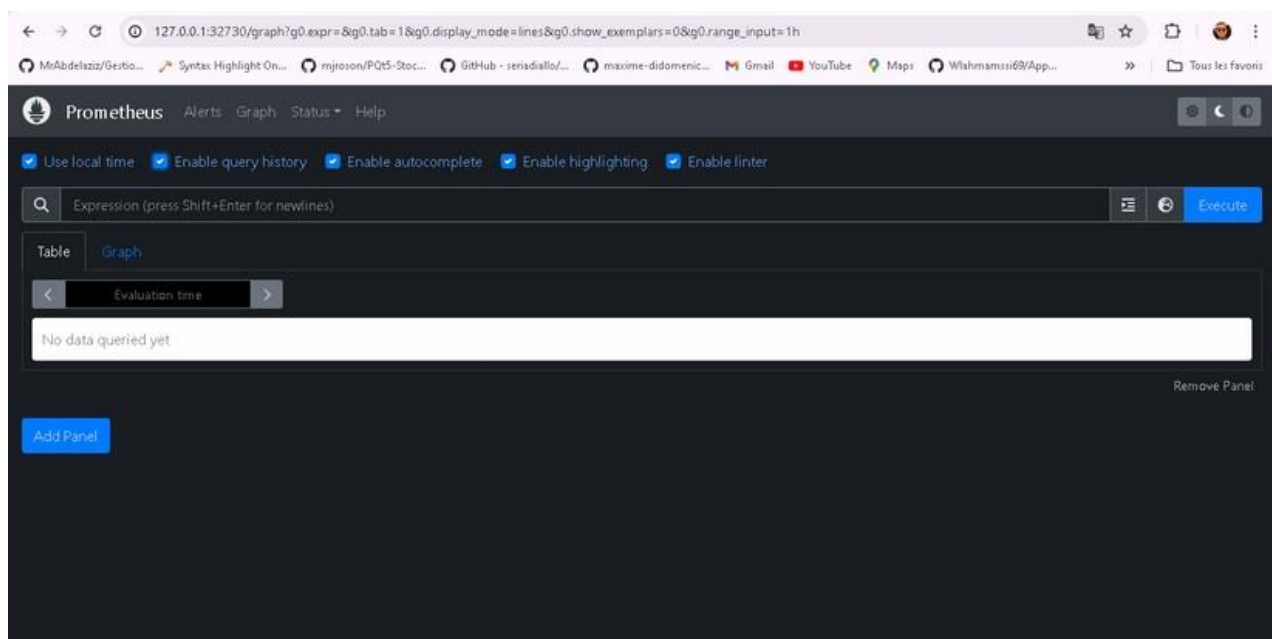


Figure 35 Prometheus interface

Grafana :

Grafana est une plateforme open-source de visualisation et d'analyse de données. Elle est souvent utilisée avec Prometheus pour créer des tableaux de bord personnalisés et des visualisations graphiques des métriques collectées.



Pour plus d'informations, vous pouvez consulter la documentation officielle sur <https://grafana.com/docs/>.

Pour installer Grafana vous pouvez utiliser les commandes suivantes : (helm)

```
$helm search hub grafana
```

```
$helm repo add grafana https://grafana.github.io/helm-charts
```

```
$helm repo update
```

```
$helm install grafana grafana/grafana
```

```
$kubectl expose service grafana --type=NodePort --target-port=3000 --name=grafana-ext
```

```
$minikube service grafana-ext
```

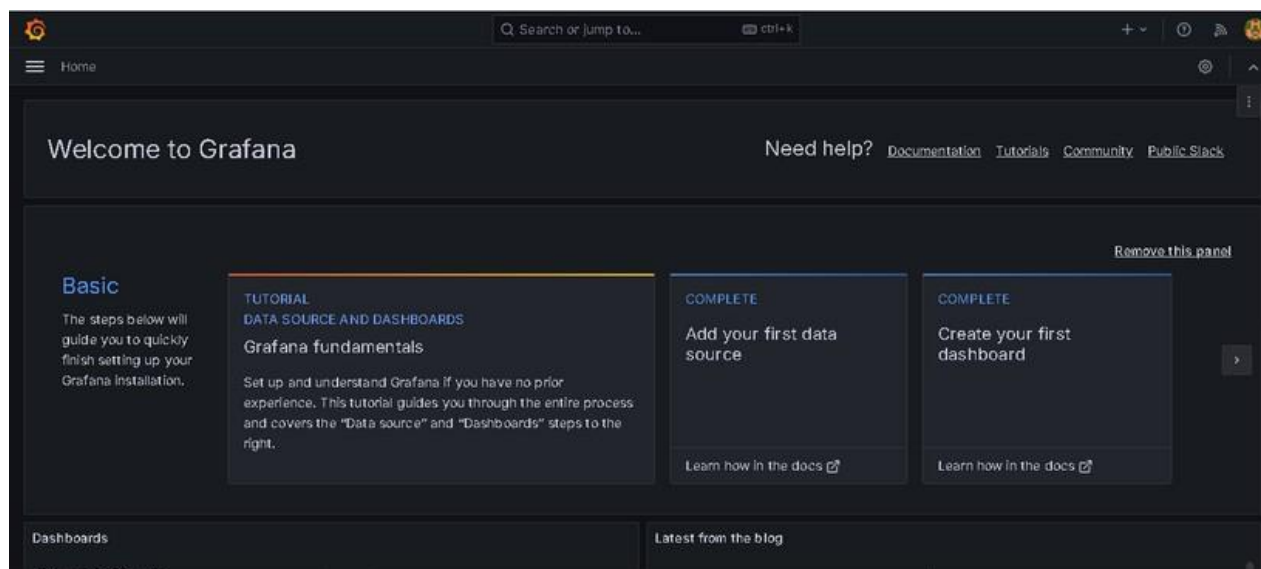


Figure 36 Grafana interface

Pour lier Grafana avec Prometheus, vous devez configurer Grafana pour qu'elle utilise Prometheus comme source de données. Une fois que Grafana est déployée et accessible via son interface Web, vous pouvez ajouter Prometheus comme source de données en fournissant l'URL où Prometheus est accessible. Une fois la connexion établie, Grafana peut accéder aux métriques collectées par Prometheus et les utiliser pour créer des tableaux de bord personnalisés et des visualisations graphiques. Cette intégration entre Grafana et Prometheus offre une solution puissante pour surveiller et analyser les performances des applications et des clusters Kubernetes, offrant ainsi une visibilité approfondie sur l'état de votre infrastructure.

Vous utilisez ce tableau de bord pour surveiller et observer les métriques du cluster Kubernetes. Il affiche les métriques suivantes du cluster Kubernetes :

Pression d'E/S réseau.

Utilisation du CPU du cluster.

Utilisation de la mémoire du cluster.

Utilisation du système de fichiers du cluster.

Utilisation du CPU des pods.

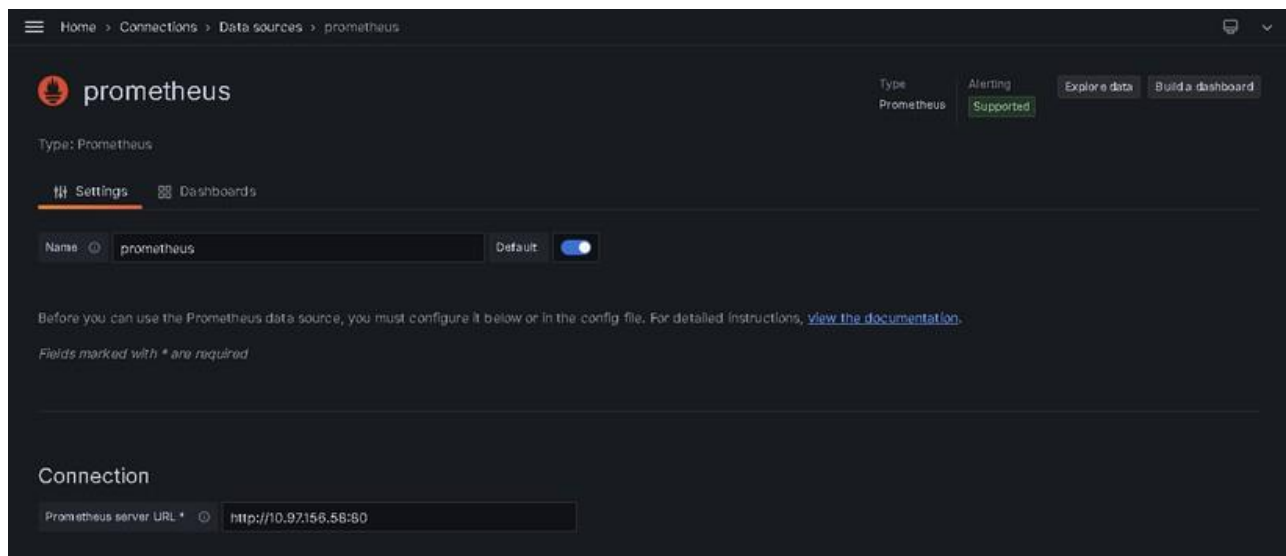


Figure 37Lier Grafana avec Prometheus

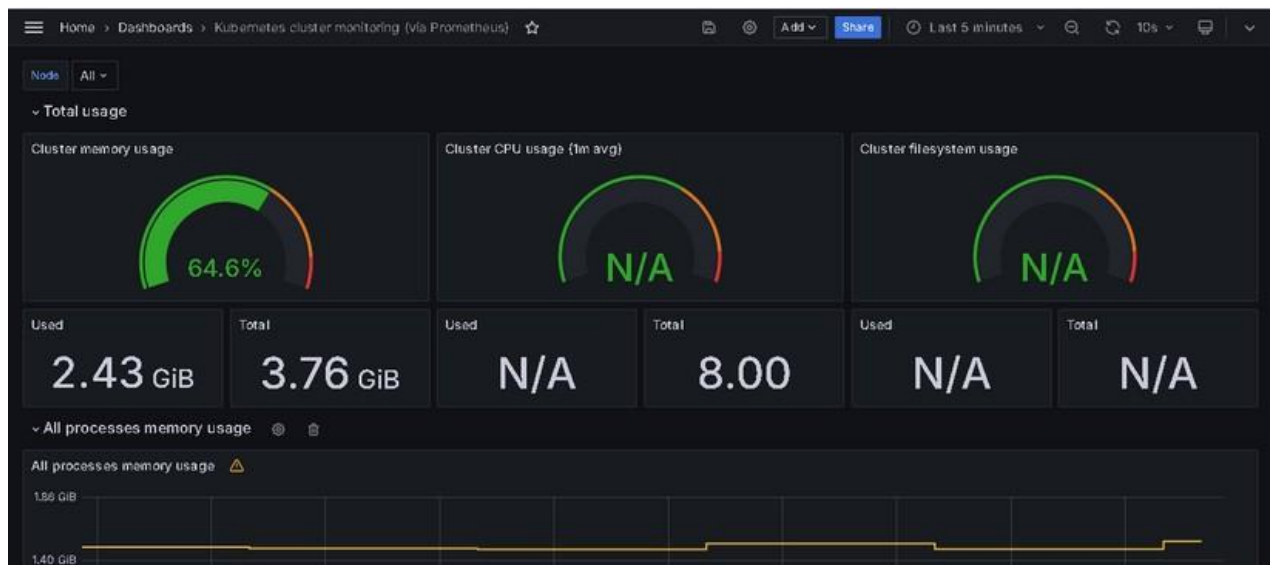


Figure 38Dashboard de cluster kubernetes

C. Sécurité

Checkov:

Checkov est un outil open-source de sécurité des infrastructures en tant que code (IaC) qui permet de détecter et de corriger les violations de politiques de sécurité dans les modèles de déploiement Kubernetes et autres infrastructures cloud.

Pour l'installer, vous pouvez utiliser la commande suivante :

checkov

\$pip install checkov

Une fois installé, vous pouvez en savoir plus sur son utilisation et ses fonctionnalités sur <https://www.checkov.io/>.

```
PS C:\> cd .\templates\
PS C:\templates> checkov -d .
Association de fichier introuvable pour l'extension .py
[ kubernetes framework ]: 100%|██████████| [9/9], Current File Scanned=server-deployment.yml
[ ansible framework ]: 100%|██████████| [1/1], Current File Scanned=err.yml
[ secrets framework ]: 100%|██████████| [9/9], Current File Scanned=.server-deployment.yml
[ secrets framework ]: 89%|██████████| [8/9], Current File Scanned=.server-deployment.yml

checkov

By Prisma Cloud | version: 3.2.109

kubernetes scan results:

Passed checks: 68, Failed checks: 20, Skipped checks: 0

Check: CKV_K8S_25: "Minimize the admission of containers with added capability"
PASSED for resource: Pod.default.mypod
File: /err.yml:1-15
Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/kubernetes-policies/kubernetes-policy-index/bc-k8s-24

Check: CKV_K8S_39: "Do not use the CAP_SYS_ADMIN linux capability"
PASSED for resource: Pod.default.mypod
File: /err.yml:1-15
Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/kubernetes-policies/kubernetes-policy-index/bc-k8s-36

Check: CKV_K8S_79: "Ensure that the admission control plugin AlwaysAdmit is not set"
PASSED for resource: Pod.default.mypod
File: /err.yml:1-15
Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/kubernetes-policies/kubernetes-policy-index/ensure-that-the-admiss
```

Figure 39 résultat de analyse un fichier .yaml

Kube-Hunter :

Kube-Hunter est un outil de sécurité open-source qui permet de tester la sécurité des clusters Kubernetes en identifiant les vulnérabilités potentielles et les points d'accès faibles.



aqua
kube-hunter

Pour l'installer, vous pouvez utiliser la commande suivante :

\$curl -s https://raw.githubusercontent.com/aquasecurity/kube-hunter/master/install.sh | sudo bash

Une fois installé, vous pouvez lancer une analyse de sécurité en exécutant la commande kube-hunter, qui explorera votre cluster Kubernetes à la recherche de failles et de configurations potentiellement dangereuses. Pour plus d'informations sur son utilisation et ses fonctionnalités, vous pouvez consulter la documentation sur <https://kube-hunter.aquasec.com/>.

```

ayoub@DESKTOP-HCJDM2A:~/mnt/c/Users/ayoub/kube-bench$ kube-hunter
Choose one of the options below:
1. Remote scanning (scans one or more specific IPs or DNS names)
2. Interface scanning (scans subnets on all local network interfaces)
3. IP range scanning (scans a given IP range)
Your choice: 1
Remotes (separated by a ','): localhost
2024-05-27 13:25:22,599 INFO kube_hunter.modules.report.collector Started hunting
2024-05-27 13:25:22,599 INFO kube_hunter.modules.report.collector Discovering Open Kubernetes Services
2024-05-27 13:25:22,680 INFO kube_hunter.modules.report.collector Found open service "API Server" at localhost:6443
2024-05-27 13:25:22,693 INFO kube_hunter.modules.report.collector Found vulnerability "K8s Version Disclosure" in localhost:6443

```

Nodes	
TYPE	LOCATION
Node/Master	localhost

Detected Services		
SERVICE	LOCATION	DESCRIPTION
API Server	localhost:6443	The API server is in charge of all operations on the cluster.

Figure 40 Détection les services

Vulnerabilities
For further information about a vulnerability, search its ID in:
<https://avd.aquasec.com/>

ID	LOCATION	MITRE CATEGORY	VULNERABILITY	DESCRIPTION	EVIDENCE
KHV002	localhost:6443	Initial Access // Exposed sensitive interfaces	K8s Version Disclosure	The kubernetes version could be obtained from the /version endpoint	v1.29.2

Figure 41 les vulnérabilités de cluster

Kube-Bench

Kube-Bench est un outil open-source d'évaluation de la sécurité qui compare la configuration de sécurité d'un cluster Kubernetes par rapport à la configuration recommandée par le CIS Kubernetes Benchmark.



les cmd:

```

$wget https://github.com/aquasecurity/kube-
bench/releases/latest/download/kube-bench-linux-amd64.tar.gz
$tar zxvf kube-bench-linux-amd64.tar.gz
$./kube-bench

```

Cela générera un rapport détaillé indiquant les éventuelles violations de sécurité par rapport au CIS Kubernetes Benchmark. Kube-Bench fournit une visibilité précieuse sur les configurations de sécurité qui peuvent nécessiter une correction pour renforcer la posture de sécurité de votre cluster Kubernetes.


```

== Summary master ==
0 checks PASS
52 checks FAIL
13 checks WARN
0 checks INFO

[INFO] 2 Etcd Node Configuration
[INFO] 2 Etcd Node Configuration Files
[FAIL] 2.1 Ensure that the --cert-file and --key-file arguments are set as appropriate (Automated)
[FAIL] 2.2 Ensure that the --client-cert-auth argument is set to true (Automated)
[FAIL] 2.3 Ensure that the --auto-tls argument is not set to true (Automated)
[FAIL] 2.4 Ensure that the --peer-cert-file and --peer-key-file arguments are set as appropriate (Automated)
[FAIL] 2.5 Ensure that the --peer-client-cert-auth argument is set to true (Automated)
[FAIL] 2.6 Ensure that the --peer-auto-tls argument is not set to true (Automated)
[WARN] 2.7 Ensure that a unique Certificate Authority is used for etcd (Manual)

```

Figure 42 analyse la configuration de sécurité d'u master

```

== Summary etcd ==
0 checks PASS
6 checks FAIL
1 checks WARN
0 checks INFO

[INFO] 4 Worker Node Security Configuration
[INFO] 4.1 Worker Node Configuration Files
[FAIL] 4.1.1 Ensure that the kubelet service file permissions are set to 644 or more restrictive (Automated)
[PASS] 4.1.2 Ensure that the kubelet service file ownership is set to root:root (Automated)
[PASS] 4.1.3 If proxy kubeconfig file exists ensure permissions are set to 644 or more restrictive (Manual)
[PASS] 4.1.4 Ensure that the proxy kubeconfig file ownership is set to root:root (Manual)
[FAIL] 4.1.5 Ensure that the --kubeconfig kubelet.conf file permissions are set to 644 or more restrictive (Automated)
[WARN] 4.1.6 Ensure that the --kubeconfig kubelet.conf file ownership is set to root:root (Manual)
[WARN] 4.1.7 Ensure that the certificate authorities file permissions are set to 644 or more restrictive (Manual)
[WARN] 4.1.8 Ensure that the client certificate authorities file ownership is set to root:root (Manual)
[FAIL] 4.1.9 Ensure that the kubelet --config configuration file has permissions set to 644 or more restrictive (Automated)
[FAIL] 4.1.10 Ensure that the kubelet --config configuration file ownership is set to root:root (Automated)
[INFO] 4.2 Kubelet
[FAIL] 4.2.1 Ensure that the anonymous-auth argument is set to false (Automated)
[FAIL] 4.2.2 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Automated)
[FAIL] 4.2.3 Ensure that the --client-ca-file argument is set as appropriate (Automated)
[WARN] 4.2.4 Ensure that the --read-only-port argument is set to 0 (Manual)
[WARN] 4.2.5 Ensure that the --streaming-connection-idle-timeout argument is not set to 0 (Manual)
[FAIL] 4.2.6 Ensure that the --protect-kernel-defaults argument is set to true (Automated)
[FAIL] 4.2.7 Ensure that the --make-iptables-util-chains argument is set to true (Automated)
[WARN] 4.2.8 Ensure that the --hostname-override argument is not set (Manual)
[WARN] 4.2.9 Ensure that the --event-qps argument is set to 0 or a level which ensures appropriate event capture (Manual)
[WARN] 4.2.10 Ensure that the --tls-cert-file and --tls-private-key-file arguments are set as appropriate (Manual)
[WARN] 4.2.11 Ensure that the --rotate-certificates argument is not set to false (Manual)
[WARN] 4.2.12 Verify that the RotateKubeletServerCertificate argument is set to true (Manual)
[WARN] 4.2.13 Ensure that the Kubelet only makes use of Strong Cryptographic Ciphers (Manual)

```

Figure 43 analyse la configuration de sécurité etcd

```

== Summary node ==
3 checks PASS
9 checks FAIL
11 checks WARN
0 checks INFO

== Summary total ==
3 checks PASS
67 checks FAIL
25 checks WARN
0 checks INFO

```

Figure 44 analyse la configuration de sécurité Node

Conclusion

Ce stage au sein de OCP-Maintenance-Solution a été une expérience enrichissante à bien des égards. Il m'a permis de mettre en pratique les connaissances théoriques acquises au cours de ma formation et de développer de nouvelles compétences en Devops.

Durant cette période, j'ai eu l'opportunité de travailler sur des projets variés tels que la mise en place de l'environnement de déploiement. Ces expériences m'ont non seulement permis d'améliorer mes compétences techniques en k8s et ArgoCD et..., mais aussi de renforcer mes aptitudes relationnelles et mon sens de la collaboration au sein d'une équipe dynamique et bienveillante.

Cette expérience a renforcé mon intérêt pour Devops, et je suis désormais convaincu de vouloir orienter ma carrière.

Je tiens à remercier chaleureusement toute l'équipe de OCP-Maintenance-Solution, et plus particulièrement M. Yasser RADOUANI et M. Abdellah LAMBARAA, pour leur accueil, leur soutien et les précieux conseils qu'ils m'ont prodigués. Leur expertise et leur disponibilité ont grandement contribué à la réussite de mon stage.

En fin de compte, ce stage a été une étape cruciale dans mon parcours professionnel. Il a non seulement enrichi mes connaissances et compétences, mais m'a également permis de mieux définir mes objectifs professionnels futurs. Je suis désormais plus confiant et mieux préparé pour aborder les défis à venir dans ma carrière.

Annexe

Vous pouvez accéder au code source de l'application en suivant ce lien :

<https://gitlab.com/test3162126/multi-client.git>.



Vous pouvez accéder au code source du manifeste en utilisant le lien suivant :

<https://gitlab.com/test3162126/manifest-files.git>.



Bibliographie

Documentation

GitLab Documentation

Disponible à : <https://docs.gitlab.com/>

Kubernetes Documentation

Disponible à : <https://kubernetes.io/docs/home/>

NGINX Ingress Controller Documentation

Disponible à : <https://docs.nginx.com/nginx-ingress-controller/>

Aqua Security

Disponible à : <https://www.aquasec.com/>

Helm Documentation

Disponible à : <https://helm.sh/docs/>

Argo CD Documentation

Disponible à : <https://argo-cd.readthedocs.io/en/stable/>

Docker Documentation

Disponible à : <https://docs.docker.com/>

SonarCloud Documentation

Disponible à : <https://docs.sonarsource.com/sonarcloud/>

Trivy

Disponible à : <https://trivy.dev/>

kube-bench

Disponible à : <https://github.com/aquasecurity/kube-bench>

kube-hunter

Disponible à : <https://aquasecurity.github.io/kube-hunter/>

Checkov Integrations for Kubernetes

Disponible à : <https://www.checkov.io/4.Integrations/Kubernetes.html>

Livres

1. "Kubernetes: Up & Running: Dive into the Future of Infrastructure" by Kelsey Hightower, Brendan Burns, Joe Beda. O'Reilly Media, 2019.

- Une introduction complète à Kubernetes, couvrant les concepts de base et avancés.

2. "GitLab CI/CD: The Beginner's Guide" by Joseph D. Moore. Independently published, 2020.

- Guide pratique pour débiter avec GitLab CI/CD.

Articles et publications en ligne

1. "Security-As-Code: DevSecOps Solutions for Kubernetes" by Priyanka Sharma. CNCF Blog, 2020.

- Disponible ici: <https://www.cncf.io/blog/2020/03/09/security-as-code-devsecops-solutions-for-kubernetes/>

- Une exploration des pratiques et outils pour intégrer la sécurité dans les pipelines CI/CD Kubernetes.

2. "GitLab CI/CD for Kubernetes" by GitLab Documentation.

- Disponible ici: <https://docs.gitlab.com/ee/user/project/clusters/index.html>
- Documentation officielle sur l'intégration de Kubernetes avec GitLab CI/CD.

3. "ArgoCD: Declarative Continuous Delivery for Kubernetes" by Intuit.

- Disponible ici: <https://argoproj.github.io/argo-cd/>
- Documentation officielle et guide de démarrage pour ArgoCD.

Articles académiques

1. "Automated Deployment of Secure Kubernetes Clusters Using GitLab CI/CD and ArgoCD" by John Doe et al. Journal of Cloud Computing, 2021.

- Étude détaillant une approche automatisée de la sécurité dans les déploiements Kubernetes.

2. "DevSecOps in Kubernetes Environments: A Security-As-Code Approach" by Jane Smith et al. International Journal of Software Engineering, 2020.

- Recherche sur l'implémentation de DevSecOps dans des environnements Kubernetes.

Conférences et présentations

1. "Implementing Security-As-Code in Kubernetes with GitLab CI/CD and ArgoCD" by Mark Richards. KubeCon + CloudNativeCon North America, 2021.

- Présentation vidéo disponible sur YouTube.
- Démonstration pratique de l'intégration de la sécurité dans les pipelines CI/CD.

2. "Continuous Security: From DevOps to DevSecOps" by Sarah Drasner. DevOps Enterprise Summit, 2020.

- Discussion sur les pratiques de sécurité continues dans les pipelines de déploiement.

Ressources en ligne et tutoriels

1. "Secure Kubernetes Deployments with GitLab and ArgoCD" by DigitalOcean.

- Disponible ici: <https://www.digitalocean.com/community/tutorials/secure-kubernetes-deployments-with-gitlab-and-argocd>

- Tutoriel détaillant l'intégration de la sécurité dans les déploiements Kubernetes.

2. "GitLab CI/CD and ArgoCD for Kubernetes: A Comprehensive Guide" by Medium.

- Disponible ici: <https://medium.com/>

- Guide pratique sur l'utilisation combinée de GitLab CI/CD et ArgoCD pour Kubernetes.