

Résumé

Afin de réduire le délai de commercialisation et produire des logiciels de qualité, Mobelite choisit d'adopter les pratiques DevOps dans les projets de ses clients.

Dans ce contexte, ce projet vise à améliorer et optimiser une solution hybride qui combiner les ressources locales de l'entreprise avec les ressources du cloud pour créer des pipelines de livraison continue qui assurent la gestion des changements, depuis le code source jusqu'à la mise en production. Il intègre l'outil Ansible pour la gestion de la configuration et l'automatisation du déploiement.

Aussi, il insère un système de recovery et de failover à base de Kubernetes et il utilise des services AWS comme Elastic Container Registry pour le repository. Toutes les images Docker sont déployées dans des conteneurs.

Après avoir effectué une étude comparative des solutions existantes, identifié les besoins et conçu l'architecture du projet, il est nécessaire de mettre en place un pipeline de livraison continue optimisé.

Ce projet a permis de minimiser le temps de déploiement dans les différents environnements et d'avoir une résilience contre les pannes d'infrastructures.

Mots clés : DevOps, Kubernetes, AWS, Ansible, Pipeline, Monitoring, cloud hybride.

Abstract

To reduce time-to-market and produce quality software, Mobelite has chosen to adopt DevOps practices in its clients projects.

In this context, this project aims to improve and optimize a Hybrid cloud solution that combine the local ressources and the cloud for continuous delivery pipeline that handles changes from source code to production. It integrates the Ansible tool for configuration management and deployment automation. Additionally, it incorporates a recovery and failover system based on Kubernetes and utilizes AWS services like Elastic Container Registry for storing all Docker images in containers.

After conducting a comparative study of existing solutions, identifying the needs, and designing the project's architecture, it became necessary to implement an optimized continuous delivery pipeline.

This project has minimized deployment time in different environments and provided resilience against infrastructure failures.

Keywords : DevOps, Kubernetes, AWS, Ansible, Pipeline, Monitoring, Hybrid cloud.

Liste des abréviations

AWS Amazon Web Services

VPC Virtual Private Cloud

EC2 Elastic Compute Cloud

IAM Identity and Access Management

EKS Elastic Kubernetes Service

ECR Amazon Elastic Container Registry

ELB Elastic Load Balancing

ALB Application Load Balancer

CI/CD Continuous integration and continuous deployment

K8s Kubernetes

DevOps Développement logiciel (dev) et de l'administration des infrastructures informatiques (ops)

Table des matières

Table des figures

Liste des tableaux

Introduction générale

La montée en puissance de l'entreprise numérique nécessite une profonde révision des méthodes de développement d'applications. Il n'est plus viable d'attendre six mois pour obtenir les livrables de développement. Avec les exigences de rapidité sur le marché et l'évolution des projets dans des configurations logicielles et d'infrastructure de plus en plus complexes, qui entraînent des risques opérationnels et de planification, il est devenu essentiel d'industrialiser les tests et de simplifier les déploiements en production. En rapprochant les équipes de développement, de test et d'exploitation, le DevOps répond précisément à ce défi numérique. Les entreprises en sont maintenant pleinement conscientes.

Parmi les pratiques couramment utilisées en DevOps, on retrouve la livraison continue et la gestion de la configuration. La livraison continue est une approche logicielle qui permet aux organisations de fournir rapidement et efficacement de nouvelles fonctionnalités aux utilisateurs. L'idée fondamentale de la livraison continue est de créer un processus reproductible et fiable d'amélioration progressive pour faire passer le logiciel du concept au client. L'objectif de la livraison continue est de permettre un flux constant de changements vers la production en utilisant un pipeline automatisé de logiciels. C'est ce pipeline de livraison continue qui rend tout cela possible.

Ce projet s'inscrit dans cette perspective en cherchant à créer un espace partagé entre une infrastructure Cloud et un Datacenter local de livraison continue en intégrant Ansible ,Kubernetes et des services AWS pour créer des conteneurs pour le monitoring. Mobelite a entrepris ce projet afin de résoudre plusieurs de ses problématiques.

Ce rapport présente les principales réalisations effectuées au cours de ce projet et est structuré en quatre chapitres :

Dans le premier chapitre,nous présentons l'entreprise d'accueil Mobelite , après nous explorons la problématique de notre sujet.Puis nous présentons les notions de base comme microservices ,DevOps et cloud. Ensuite, nous allons définir la méthodologie de développement utilisée .Enfin, nous allons présenter la planifications des sprints.

Dans le deuxième chapitre, nous présentons une étude comparative des solutions existantes les plus répandues sur le marché et les plus adéquate, ainsi que l'outil choisi dans chaque solution.

Dans le troisième chapitre, analyse et conception, nous exposerons les besoins

fonctionnels et non fonctionnels, les acteurs, les cas d'utilisations et l'architecture physique globale.

Le dernier chapitre est consacré pour présenter les tâches réalisées pour implémenter notre projet.

Chapitre 1

Cadre général du projet et étude de l'existant

Introduction

Ce chapitre est consacré à la présentation du cadre général de notre projet. En premier lieu, nous présentons la société d'accueil. Par la suite, nous présentons le cadre du projet qui explique la problématique . Après, on définit par une présentation de la méthodologie que nous allons adopter pour le développement de notre projet. On finit par la présentation des solutions existantes sur le marché avec une étude comparative de ces dernières et présentation de la solution proposée.

1.1 Société d'accueil

Mobelite est une agence spécialisée dans le conseil en stratégie mobile, la conception et le développement d'applications mobiles , de sites web et le marketing mobile. Mobelite est forte d'une équipe d'experts dans la réalisation d'applications mobiles sur les plateformes les plus répandues et les applications Web. Mobelite dispose d'équipes commerciales et marketing à Paris (France), et d'équipes de développement à Tunis et Monastir (Tunisie).

Ainsi, Mobelite est une société experte dans la création des sites web intuitifs et ergonomiques pour tous les supports soit desktop, tablette ou mobile et avec les plus récentes technologies et Framework de développement comme React JS, Node JS et Angular. Les équipes de mobelite maîtrisent parfaitement la conception et le développement d'applications natives iOS et Android pour tous les différents supports que ce soit smartphone, tablette, Watch et TV. Mobelite excelle dans le conseil de ses clients dans différentes parties de projets comme l'analyse des besoins, UX/UI, la conception, le design, le développement, SEO, DevOps et l'hébergement. Tout ça est réalisé selon la méthodologie Agile, afin de maximiser les possibilités d'itérations sur le concept, et d'introduire plus de flexibilité sur les arbitrages fonctionnels à faire(voir figure 1.2).

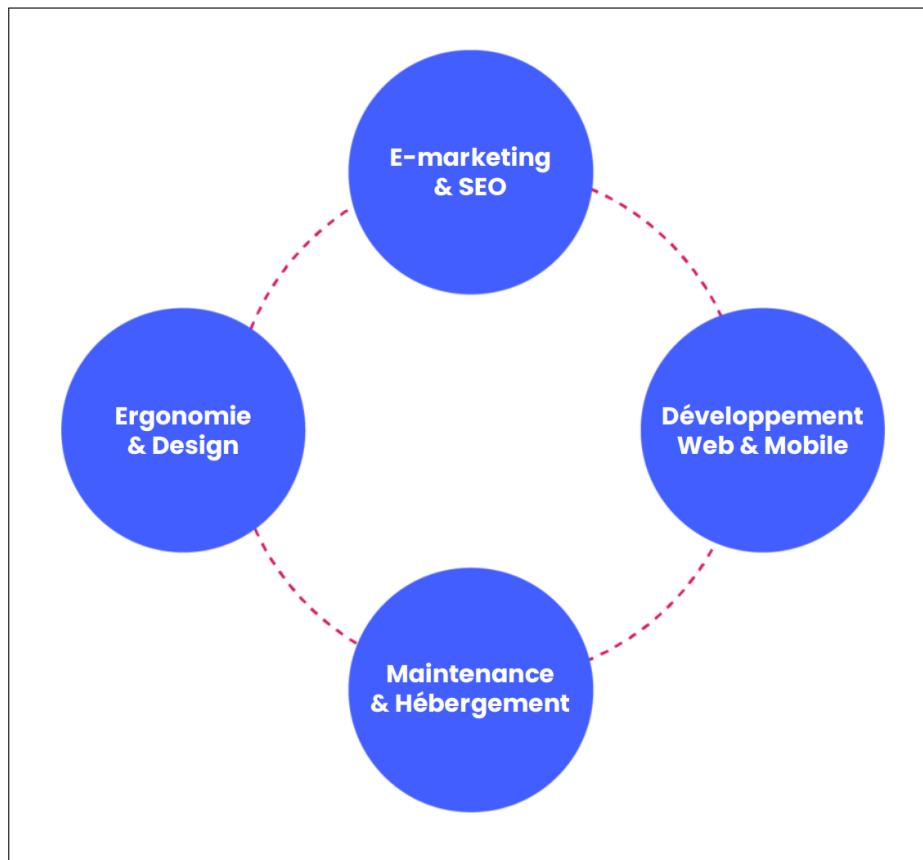


FIGURE 1.1 – différents domaines de mobelite

1.2 Problématique

Le déploiement des systèmes de gestion de code source tells que Nexus, SonarQube ou le Monitoring sur un seul serveur local présente plusieurs problèmes qui affectent la performance de serveur et qui rendent l'exécution des différentes applications ou l'ajout des nouvelles fonctionnalités plus difficile . Aussi, la maintenance de plusieurs applications en même temps sera compliquée et nécessite une planification minutieuse pour éviter l'interruption des processus d'autres applications. Les besoins d'entreprise changent au cours des temps et la capacité d'un seul serveur sera insuffisante pour rependre à la charge des données. Les mise à jour des applications peuvent impacter d'autres applications à cause de limites des ressources. En terme de sécurité, une attaque sur le serveur cause une perte des données très grande que sera difficile de récupérer en conséquence de l'utilisation d'un seul serveur. C'est dans ce cadre que la société souhaite créer son propre solution .

1.3 Notions de base

Dans ce qui suit,nous présentons les notions de base que nous utiliserons pour réaliser le projet.

1.3.1 Microservice

Avant l'apparition de l'architecture micro-service, les applications sont construites de manière monolithique, qui est considérée aujourd'hui comme méthode impertinente. L'utilisation de l'architecture monolithique, rend l'application très volumineuse avec des difficultés à enrichir les fonctions et le traitement des problèmes. Les microservices créent une application unique à partir de plusieurs petits services reliés de manière flexible. Ces services ont leur propre logique et leur propre base de données pour un usage spécifique. Chaque service peut être développé, mis à jour, déployé et mis à l'échelle sans affecter les autres services. Les mises à jour logicielles peuvent être plus fréquentes, ce qui améliore la fiabilité, la disponibilité et les performances(Voir figure 1.3).

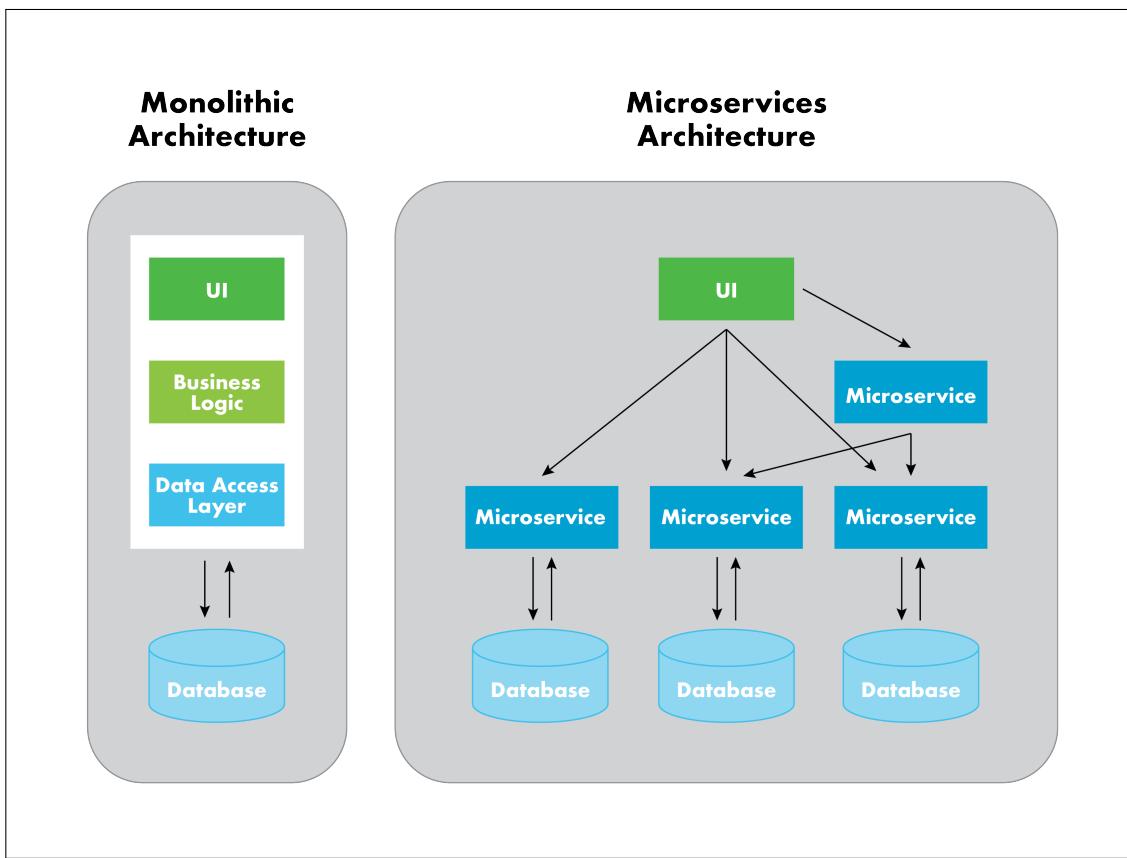


FIGURE 1.2 – Architecture monolithique vs Micro services

1.3.2 DevOps

Combinant développement (Dev) et opérations (Ops), DevOps est l'union des personnes, des processus et des technologies destinés à fournir continuellement de la valeur aux clients. DevOps permet la coordination et la collaboration des rôles autrefois cloisonnés (développement, opérations informatiques, ingénierie qualité et sécurité) pour créer des produits plus performantes et plus fiables. En adoptant une culture DevOps ,ainsi que des pratiques et outils DevOps, les équipes peuvent mieux répondre aux besoins des clients, accroître la confiance suscitée par les

applications qu'elles développent, et atteindre plus rapidement les objectifs de leur entreprise [?] (voir figure 1.4).

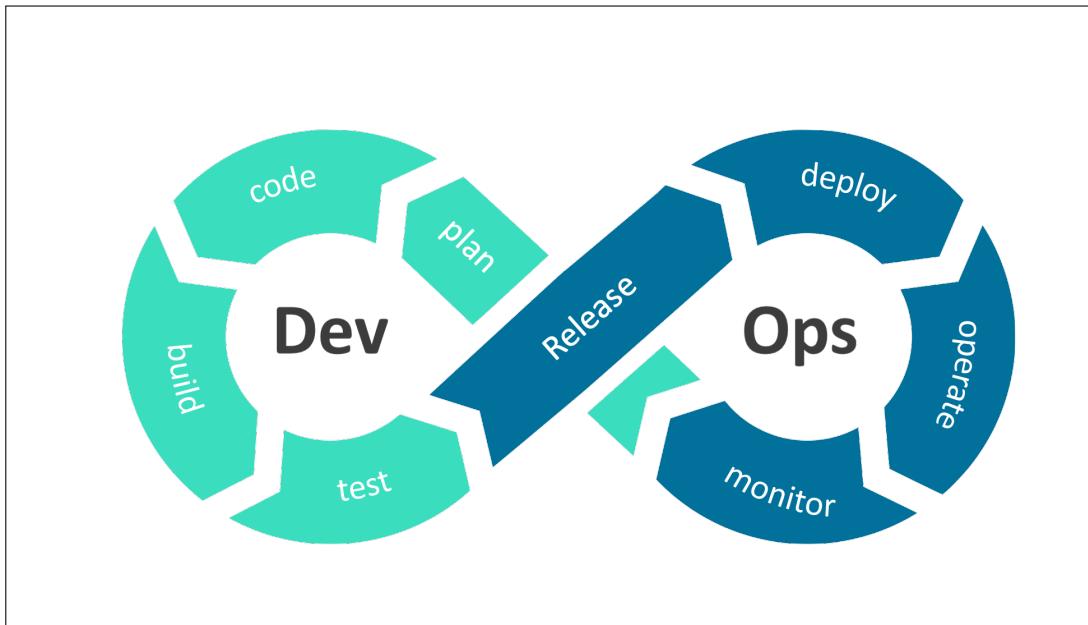


FIGURE 1.3 – Cycle de vie DevOps

1.3.3 Cloud

Le cloud n'est pas une entité physique, mais un vaste réseau de serveurs distants éparpillés tout autour de la planète, reliés entre eux, et destinés à fonctionner comme un écosystème unique. Ces serveurs sont conçus pour stocker et gérer des données, exécuter des applications, ou fournir du contenu ou des services (vidéos diffusées en continu, courrier web, logiciels bureautiques de productivité et autres réseaux sociaux). Au lieu d'accéder à des fichiers et données stockées sur un ordinateur local ou personnel, vous accédez à ces ressources en ligne à partir de n'importe quel appareil compatible avec Internet : les informations sont disponibles en tout lieu et tout le temps.[?]

1.3.3.1 Modèles de services

Il existe 3 modèles de services sur le cloud :

-Software as a Service (SaaS) : Le Software as a Service, également connu sous le nom de SaaS, est un service basé sur le cloud où, au lieu de télécharger un logiciel que votre PC de bureau ou votre réseau professionnel peut exécuter et mettre à jour, vous accédez à une application via un navigateur internet. L'application logicielle peut être un logiciel de bureautique ou de communication unifiée parmi un large éventail d'autres applications professionnelles disponibles[?] (voir figure 1.4).

-Platform as a Service (PaaS) : La Platform-as-a-service (PaaS) est un type d'offre de cloud computing dans lequel un fournisseur de services fournit une plateforme à ses clients, leur permettant de développer, d'exécuter et de gérer des applications commerciales sans avoir à construire et à maintenir l'infrastructure que ces processus de développement de logiciels requièrent généralement(Voir figure 1.4)[?].

-Infrastructure as a Service (IaaS) : Infrastructure as a service (IaaS) est un type de service de cloud computing qui offre des ressources de calcul, de stockage et de mise en réseau essentielles à la demande, et sur une base de paiement à l'utilisation(Voir figure 1.4)[?].

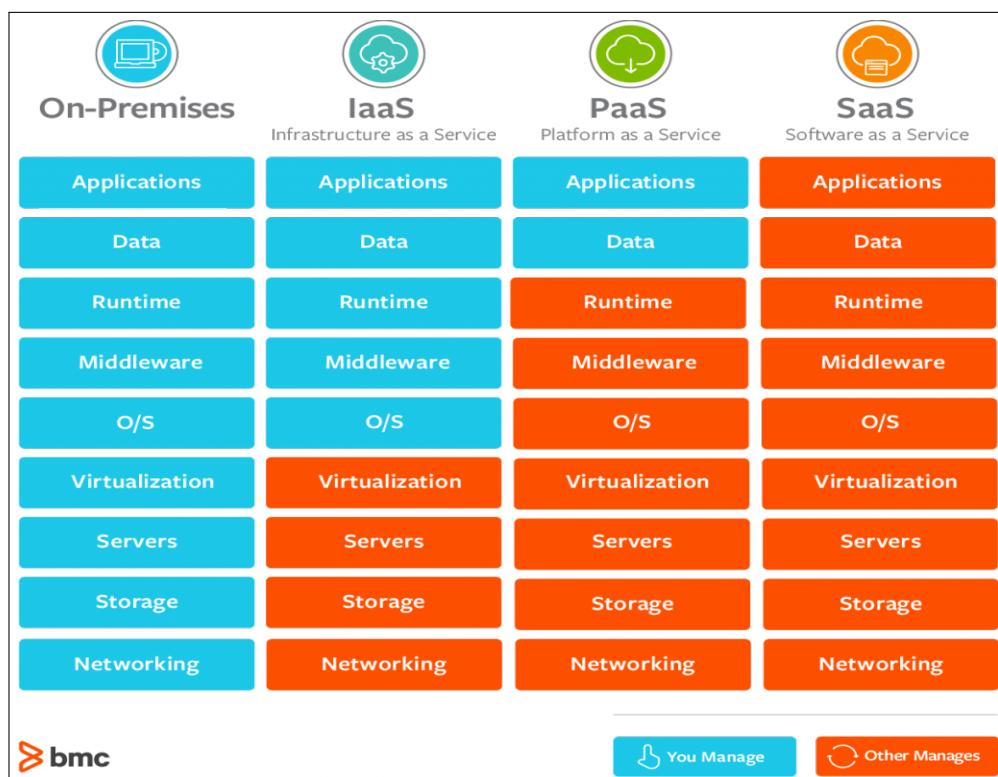


FIGURE 1.4 – Différents architecture entre SaaS ,PaaS et IaaS

1.3.3.2 Les modèles de déploiement

Il existe 3 modèles de déploiement sur le cloud :

-Cloud Privé : Le cloud privé est un modèle informatique qui offre un environnement propriétaire dédié à une seule entité commerciale. Comme les autres types d'environnements du cloud computing, le cloud privé fournit des ressources informatiques étendues et virtualisées via des composants physiques stockés sur place ou dans le centre de données d'un fournisseur[?].

-Cloud Public : Le cloud public est un type de calcul dans lequel les ressources sont proposées par un fournisseur tiers via Internet, et sont partagées par les or-

ganisations et les individus qui souhaitent les utiliser ou les acheter [?].

-Cloud Hybride : Un cloud hybride est un environnement informatique mixte dans lequel des applications s'exécutent à l'aide d'une combinaison de ressources de calcul, de stockage et de services dans différents environnements (clouds publics et clouds privés, y compris des centres de données sur site ou en périphérie)[?](voir figure 1.5).

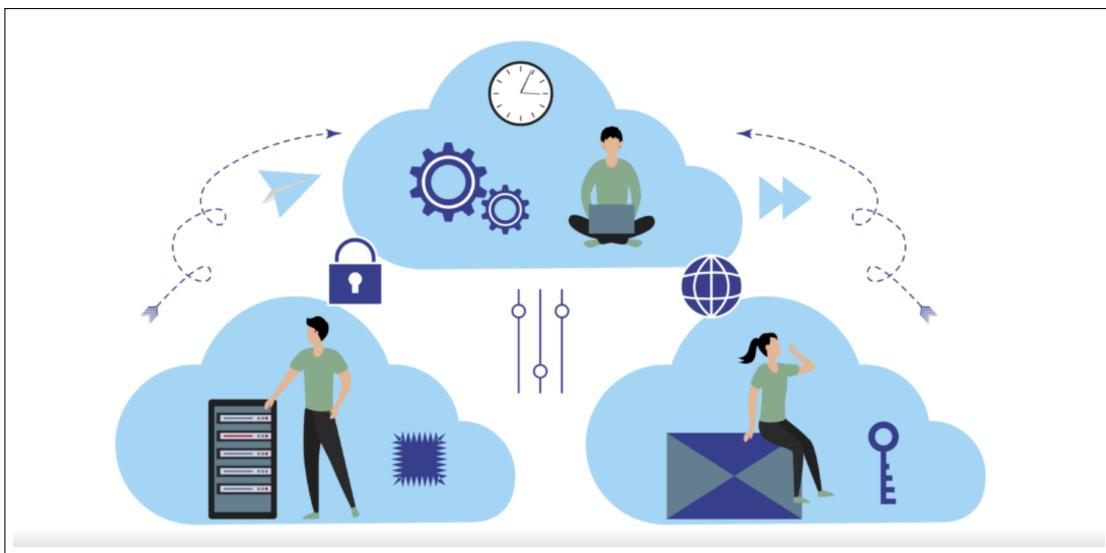


FIGURE 1.5 – Différents type du cloud

1.4 Méthodologie de développement adoptée

Avant de réaliser un projet informatique, il est essentiel de sélectionner une démarche de travail et une procédure de suivi pour obtenir un logiciel stable. Il s'agit d'un cadre permettant de structurer, de planifier et de contrôler le développement d'une application.

Pour la réalisation de notre projet nous avons opté pour la méthodologie agile SCRUM car elle améliorer la flexibilité de projet et réduit le temps de livraison de produits complet au client. En effet , la plupart des projets réalisés dans l'entreprise appliquent cette méthodologie.

1.4.1 Méthode agile

Agile est une méthode de gestion de projet conçue pour distribuer en continu les logiciels opérationnels en fonction d'itérations rapides. Il permet aux équipes de développer progressivement un produit de qualité et d'adapter leur processus en fonction des besoins du projet.

1.4.2 Présentation de la méthodologie Scrum

Scrum est une méthodologie agile pour l'élaboration, la réalisation et le soutien de projets complexes. Elle est basée sur la division du produit sur différentes itération sprints. Scrum est plus utile lorsque les exigences sont variables et peuvent changer beaucoup de fois au cours du cycle de vie, ce qui donne donc la capacité d'avoir un projet flexible capable de traiter les changements fréquents.

1.4.3 Acteurs de la méthode agile Scrum

Méthode agile Scrum se compose par 3 acteurs ce suit :

-Scrum Master : Il s'agit de la personne chargée d'orienter l'équipe de travail vers la bonne voie, d'assurer la bonne pratique des règles de la méthode Scrum et d'organiser son équipe.

-Product Owner : il représente les clients et assure que leurs besoins et visions soient réalisés dans le projet. Il travaille généralement en collaboration directe avec l'équipe de développement.

-Equipe de développement : sont les personnes responsables de la transformation des besoins du client. Ces besoins sont définis par le Product Owner à travers des fonctions utilisables. L'équipe est composée d'au moins 3 personnes jouant les rôles de développeur ou de concepteur.

1.4.4 Évènements de la méthode agile Scrum

Scrum définit cinq types d'évènements :

-Le sprint : Chaque sprint est de durée maximale de 4 semaines pendant laquelle une version de produit est réalisée. Une fois un sprint terminé un nouveau a déjà commencé avec une liste de fonctionnalités et un objectif à réaliser.

-Planification d'un sprint : À chaque début de sprint, cette réunion est conçue pour déterminer les tâches à réaliser pendant le sprint. L'organisation de ces tâches est effectuée par l'équipe de développement et le Product Owner.

-Mélée quotidienne : C'est une réunion d'une durée de 15 minutes faite par l'équipe chaque jour pour définir l'objectif de la journée et identifier les obstacles s'il y en a quelques-uns.

-Revue de sprint : Représente le travail réalisé par l'équipe au cours du sprint et le comparer avec le produit attendu.

-Rétrospective de sprint : Le but de cet événement est de déterminer les problèmes intervenus dans la période du sprint, l'efficacité des outils et déterminer ce qui peut être amélioré.

1.4.5 Les artefacts

-Product Backlog : Il s'agit d'une liste des besoins et exigences à recueillir pour créer le produit désiré, ce document est la responsabilité de Product Owner(voir figure 1.6).

-Sprint Backlog : C'est l'ensemble des données permettant la réalisation des objectifs du sprint. Ce document est mis à jour par l'équipe de développement régulièrement.

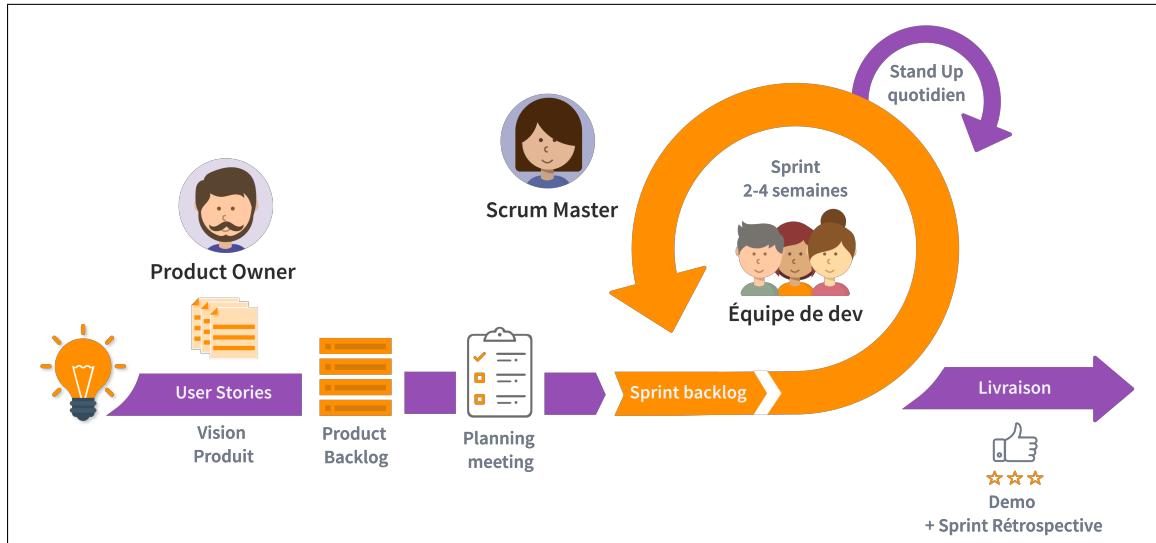


FIGURE 1.6 – Processus de méthode Scrum

1.5 Planification des sprints

Pour une meilleure optimisation du développement du projet nous avons divisé le travail en des Sprints présentés dans un diagramme de Gantt qui décrit l'état d'avancement dans le temps des différentes activités (voir figure 1.7) :

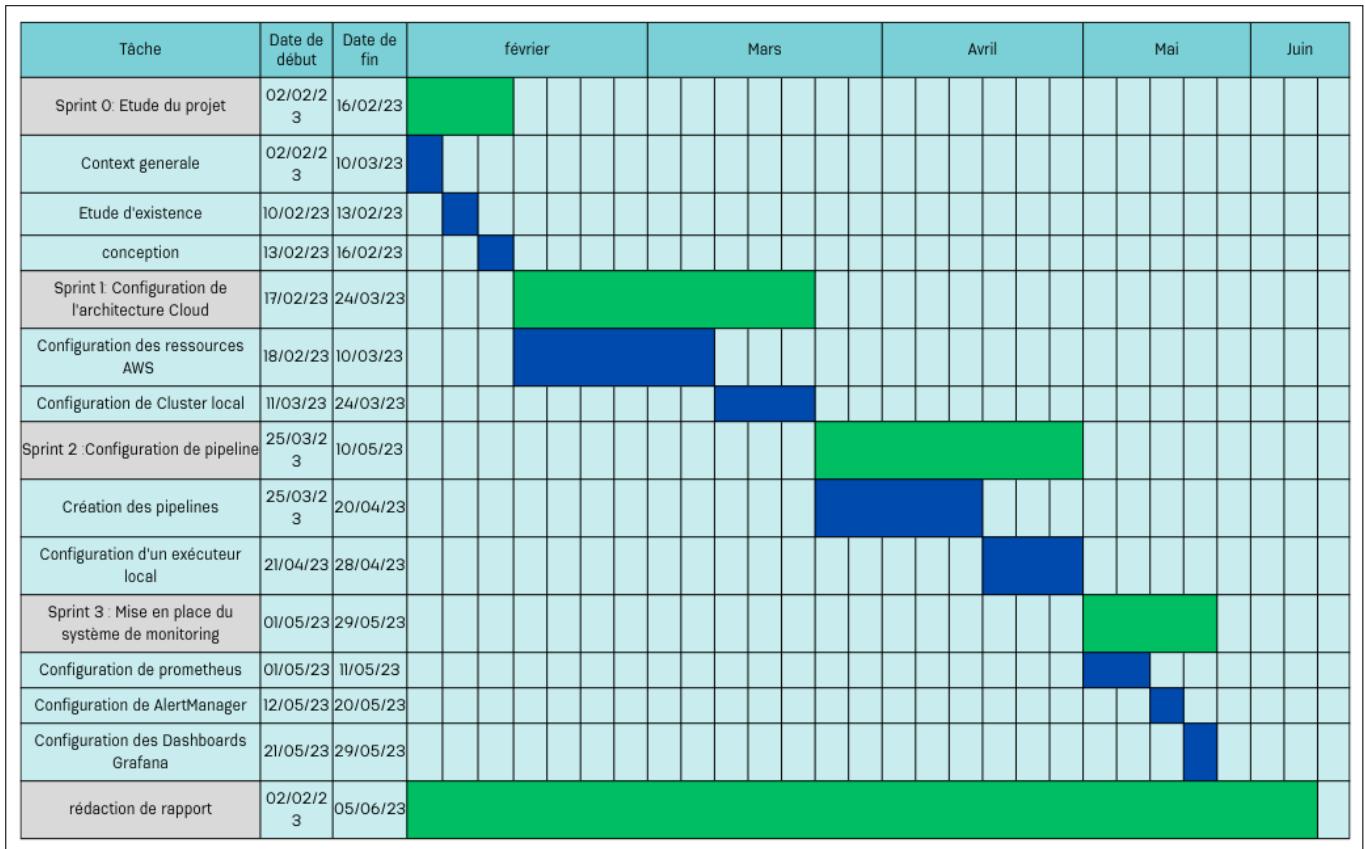


FIGURE 1.7 – Diagramme de Gantt

1.6 Les solutions existantes

On va passer en revue les solutions similaires à notre solution. Nous allons étudier les points forts ainsi que les points faibles de ces solutions afin de montrer pourquoi nous devrions développer notre propre solution. Voici une présentation de certaines solutions existantes :

1.6.1 Azure Devops

Azure DevOps prend en charge une culture collaborative et un ensemble de processus qui rassemblent les développeurs, les responsables de projets et les contributeurs pour développer des logiciels. Elle permet aux organisations de créer et d'améliorer les produits à un rythme plus rapide que possible avec les approches traditionnelles de développement de logiciels. Azure DevOps Services vous donne également accès aux serveurs de génération et de déploiement cloud et aux insights sur les applications. Démarrer gratuitement et créez une organisation. Ensuite, chargez votre code pour partager ou contrôler le code source. Commencez à suivre votre travail à l'aide de Scrum, Kanban ou d'une combinaison de méthodes [?] (voir figure 1.8). Elle a des points forts mais aussi des points faibles :

- Points forts :

– Azure DevOps offre une haute disponibilité, sécurité solide et offre de bonnes options d'évolutivité.

– Azure DevOps est un outil informatique qui adapte en fonction des exigences du projet sans problème, offre à l'utilisateur une flexibilité en matière de gestion de l'infrastructure.

- **Points faibles :**

– Azure DevOps propose de nombreuses fonctions prêtes à l'emploi, mais la personnalisation peut être limitée. Les organismes ayant des besoins spéciaux peuvent avoir de la difficulté à adapter l'outil à leurs besoins précis.

– Azure DevOps fonctionne particulièrement sur cloud qui nécessite une connexion internet pour fonctionner, l'absence d'accès hors ligne peut constituer un inconvénient dans les situations où les personnes doivent travailler à distance avec une connectivité Internet limitée.

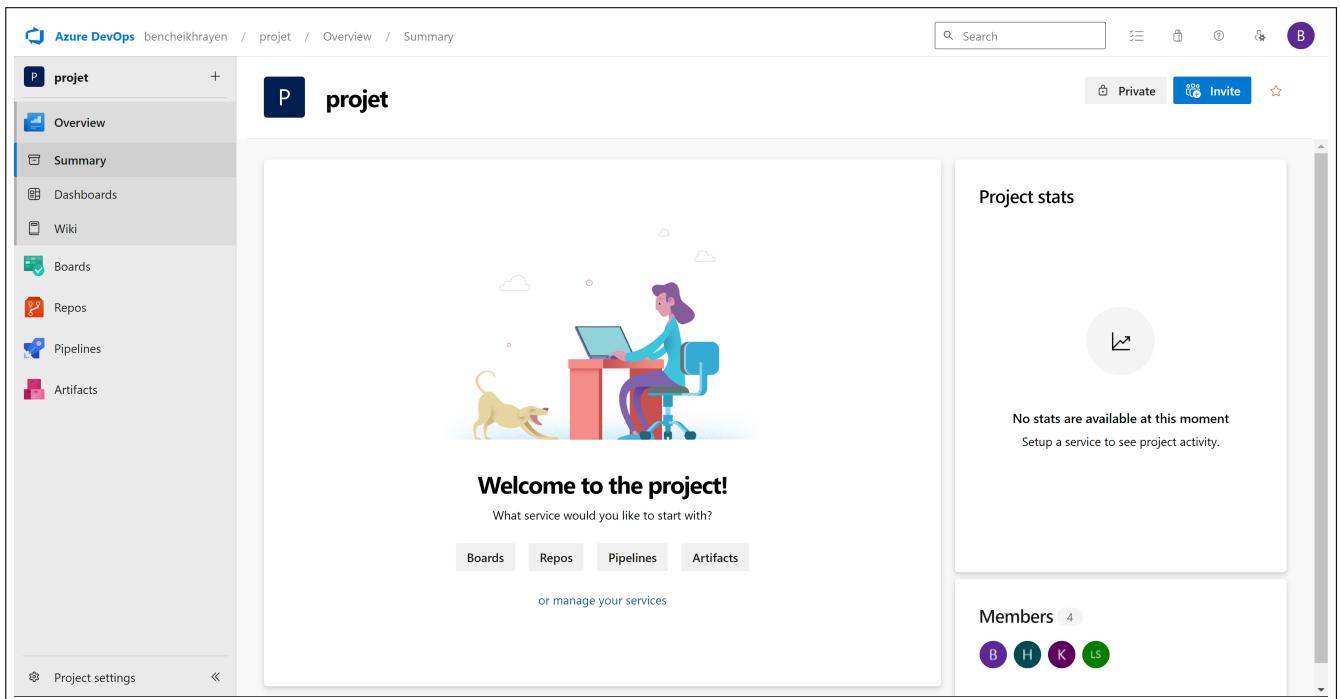


FIGURE 1.8 – Azure DevOps

1.6.2 Amazon Web Service DevOps

AWS fournit un ensemble de services flexibles, conçus pour permettre aux entreprises de créer et livrer des produits avec plus de rapidité et de fiabilité à l'aide d'AWS et des pratiques de DevOps. Ces services simplifient la mise en service et la gestion de l'infrastructure, le déploiement de code d'application, l'automatisation des processus de publication de logiciel et le suivi des performances de l'application et de l'infrastructure [?](voir figure 1.9).

Cependant, comme toute autre technologie, AWS présente sur des points forts et des points faibles :

- Points forts :

– Aws offre une grande évolutivité pour les entreprises peuvent facilement augmenter ou réduire leurs ressources en fonction de leurs besoins .Aussi il présente un niveau de fiabilité élevé et il fournit aux entreprises une gamme de services, notamment le stockage, la gestion de bases de données, la puissance de calcul et l'analyse

– AWS fournit des outils de redondance et de tolérance aux défaillances, tels que l'auto-scaling et la disponibilité de plusieurs zones. Ces caractéristiques renforcent la disponibilité des applications et assurent des performances fiables.

- Points faibles :

– AWS est complexe à mettre en place et à gérer ses ressources. Aussi, les entreprises doivent prendre des mesures pour garantir la protection de leurs données.

– L'adoption d'AWS DevOps peut créer une dépendance envers les différents services, ce qui peut mener à un verrouillage des fournisseur.

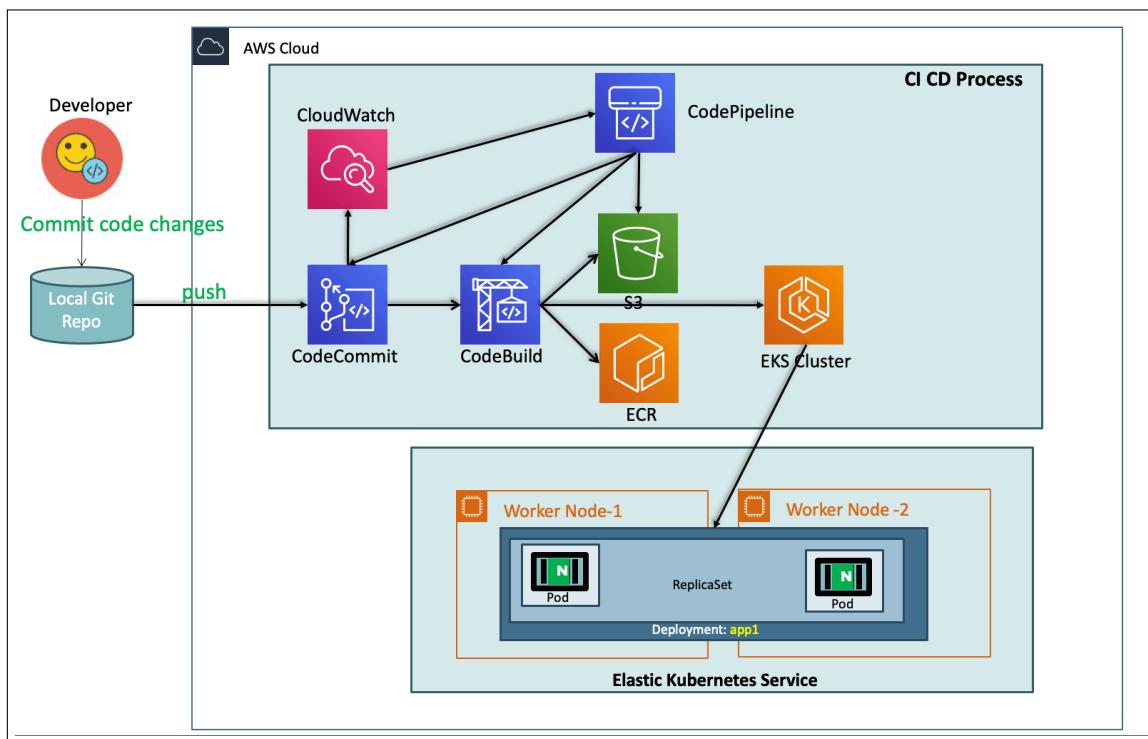


FIGURE 1.9 – Les Services AWS utilisé pour DevOps.

1.6.3 Google Cloud Platform DevOps

Google propose de nombreux services et fonctionnalités visant à aider les ingénieurs DevOps à disposer de tout ce dont ils ont besoin pour respecter les normes de qualité et de sécurité les plus élevées tout en automatisant la majorité du processus. De plus, vous travaillez avec des outils GCP DevOps efficaces qui non seulement augmentent la qualité des applications, mais améliorent également la vitesse du cycle de développement. Ces outils DevOps s'adressent directement aux ingénieurs logiciels car ils permettent une configuration rapide et simple, avec une

interface intuitive qui facilite l'utilisation efficace des outils et méthodologies de livraison continue et de déploiement continu (intégration continue/ déploiement continu(CI/CD))[?](Voir figure 1.10).

Ainsi GCP a des points forts et des points faibles :

- **Points forts :**

- GCP fournit des services d'intégration et de développement d'applications et assure une bonne sécurité.
- GCP DevOps rende facile la collaboration entre les équipes de développement et les équipes opérationnelles grâce à des services partagés.

- **Points faibles :**

- Quelque intégration peut être plus limitées avec les services et les outils de Google par rapport à des solutions locales. De plus, la configuration initiale du GCP est complexe en raison de la grande diversité de services et d'options disponibles.
- L'utilisation efficace de GCP DevOps exige un personne qualifié spécialisé dans les technologies cloud, l'automatisation et la gestion de l'infrastructure.

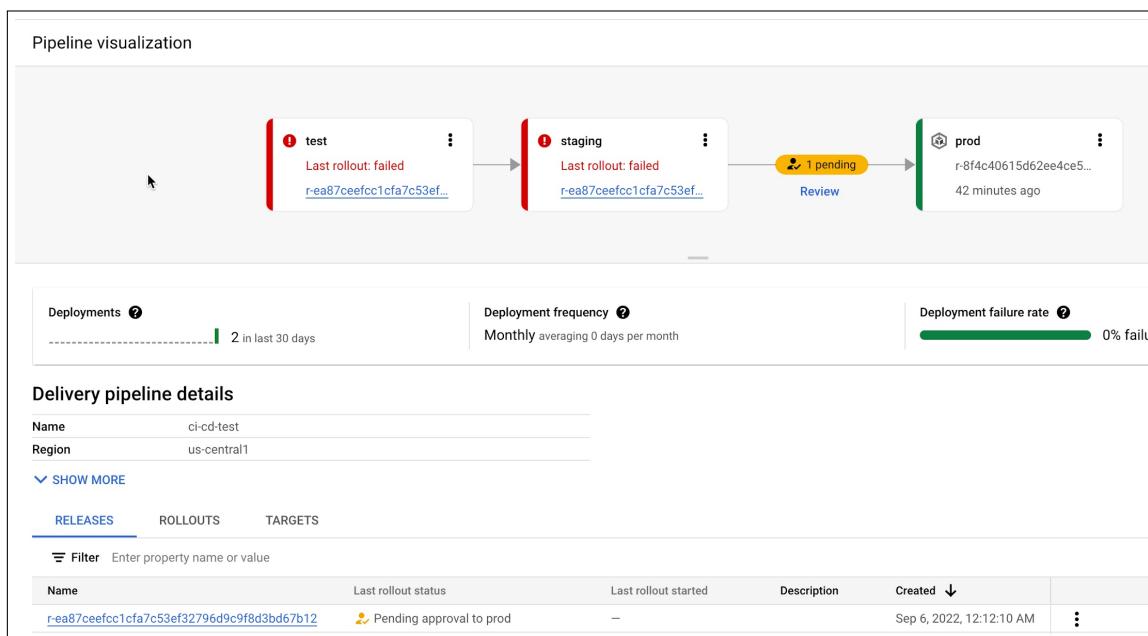


FIGURE 1.10 – Interface de GCP Pipeline.

1.7 Critique des solutions existantes

Nous allons comparer les solutions présentées ci-dessus selon plusieurs critères qu'on va définir par rapport aux besoins de l'entreprise(voir tableau 1.1).

C1 -La confidentialité des données d'application : La solution doit privilégier la confidentialité des données puisque certains clients préfèrent leur

	Azure DevOps	Amazon Web Service DevOps	Google Cloud Platform DevOps
C1	X	X	X
C2	✓	✓	✓
C3	✓	✓	X
C4	X	X	X

Légende :
 X : La solution ne conforme pas à la critère.
 ✓ : La solution conforme à la critère.

TABLE 1.1 – Tableau comparatif

application exécutée sur le serveur local de la société.

C2 -Intégration de solutions source libre : Amazon cloud, azure cloud ou toute autre solution cloud doit intégrer certaines software open source comme services à utiliser dans votre architecture cloud.Bien que les logiciels libres puissent être hautement personnalisables, le niveau de personnalisation fourni par un fournisseur de services peut être restreint.

C3 -Coût de projet : L'utilisation d'Amazon Cloud ainsi tout autre fournisseur pour déployer une application dans le Cloud semble peu coûteux au début, mais après que la charge de travail augmente, les coûts aussi augmentent. Il s'agit de trouver une solution qui profite du cloud et des avantages du logiciel libre.

C4 -Problèmes de compatibilité : La personnalisation peut présenter des problèmes de compatibilité, surtout si la personnalisation interagit avec d'autres services ou composants au cours du déploiement. Cela peut entraîner des temps d'arrêt ou d'autres problèmes de performance.

En se basant sur le tableau comparatif ci-dessus, on constate que les solutions basées uniquement sur le cloud ne sont pas utiles, et qu'une combinaison de ressources cloud et locales est conforme à notre projet.

Notons que cette évaluation ne concerne que notre situation, dans d'autres cas les solutions existantes peuvent être utile.

1.8 Solution proposée

Pour rectifier les problèmes auxquels l'équipe DevOps fait face chaque jour et afin de préparer un environnement de développement flexible, Mobelite nous a proposé de mettre en place une solution Hybride cloud qui bénéficie à la fois de certains services de cloud avec un accès privé aux données. En effet, le projet consiste à implémenter la conteneurisation qui nous offrira une virtualisation des ressources de manière légère, flexible et puissante. Le déploiement d'un kubernetes

cluster qui est un système de gestion de conteneurs qu'on va utiliser pour optimiser le déploiement des applications avec l'utilisation d'autres produits comme Ansible pour l'optimisation de cette infrastructure. La solution a des objectifs importants pour l'entreprise comme la réduction des coûts de développement et d'exploitation. Le Cycle de développement sera plus court grâce à l'automatisation et le déploiement rapide des nouveaux environnements. Aussi, la solution garantira la supervision totale et continue de la plateforme, et la haute disponibilité du système.

Conclusion

Dans ce chapitre nous avons présenté la société d'accueil, la problématique et la méthodologie de développement. Aussi, nous avons réalisé une étude de quelques solutions existantes, puis nous avons réalisé une comparaison de l'existant pour connaître les solutions compatibles avec notre sujet. Après cela, nous avons présenté la solution proposée par mobelite.

Chapitre 2

Analyse des besoins et conception

Introduction

L'objectif de l'étape de spécification et d'analyse des besoins est de déterminer les fonctionnalités différentes attendues du système. Au cours de ce chapitre, nous présentons d'abord les acteurs concernés dans notre système. Ensuite, nous allons illustrer les besoins fonctionnels et non fonctionnels. Puis, nous allons détailler le product backlog de notre projet. Ces exigences seront exprimées enfin sous forme de diagramme de cas d'utilisation qui sera détaillé par des scénarios possibles. Par la suite, nous présentons la conception de notre application qui présente une étape fondamentale qui précède la réalisation par une description des différents diagrammes de séquence, classe et déploiement. Nous procérons ensuite à la définition de la méthodologie de conception. Et nous terminons par représenter l'architecture technique de notre projet.

2.1 Acteurs

En génie logiciel et plus particulièrement en UML, un acteur est une entité qui définit le rôle joué par un utilisateur ou par un système qui interagit avec le système modélisé[?].

– Développeur : C'est l'utilisateur qui peut modifier le code source d'un objectif donné (ajouter une fonctionnalité, corriger l'erreur, etc.) puis les déposer dans Github et ensuite suivre la compilation de l'application.

– Ingénieur DevOps : Est considéré comme l'acteur principal, son rôle est la mise en place du cluster, pipeline, surveiller l'état du système et configurer son infrastructure.

– Client : C'est l'utilisateur qui peut accéder à l'application déployée sur le

cluster au moyen d'un site Web.

2.2 Besoins fonctionnels

Pour le bon fonctionnement de notre projet, il est nécessaire de définir concrètement les fonctionnalités qui seront implémentées dans le but de les rendre plus appropriées aux besoins de l'entreprise. Pour cela nous avons allons présenter les besoins fonctionnels de notre projet :

- Mettre en place une solution hybride qui permet à l'entreprise d'exécuter des applications localement tout en profitant des avantages des services cloud.
- Orchestrer les conteneurs pour garantir une haute disponibilité pour le déploiement et l'exécution des applications.
- Automatiser le processus de déploiement avec Ansible pour aider avec les charges de travail.
- Surveiller l'état du déploiement.
- Créer un répertoire pour toutes les images Docker dans ECR (Elastic Container Registry).
- Enregistrer les artefacts dans un répertoire Nexus.
- Utiliser Prometheus et Grafana pour surveiller le cluster kubernetes.
- Assurer la qualité de code source avec SonarQube.
- Gérer automatique des étapes de déploiement de clusters.

2.3 Besoins non-fonctionnels

Les besoins non fonctionnels établissent toutes les conditions nécessaires au bon fonctionnement du système et à l'amélioration de la qualité des services. Et pour répondre aux exigences fonctionnelles, notre projet doit respecter une série de propriétés contribuant à une meilleure qualité de la solution obtenue. Nous détermi-

mineron l'ensemble des contraintes à respecter pour garantir le bon déroulement du projet. Parmi les critères nous citons :

–Confidentialité : Notre solution permettra de garantir la sécurité des données, des applications et des utilisateurs.

–Performance : Rapidité du déploiement d'application sur des pods kubernetes et assurer la bonne exécution de ses applications.

texttt

–Disponibilité : Les clusters Kubernetes sont conçus pour permettre une évolutivité horizontale, ce qui signifie que les applications peuvent être déployées sur plusieurs nœuds et gérer de manière dynamique les charges de travail en fonction des besoins.

–Maintenance : La modification d'un déploiement dans un kubernetes cluster doit être facilement maintenable et adaptable à de nouvelles exigences de sorte qu'il peut y avoir un changement ou l'ajout de nouvelle fonctionnalité.

–Portabilité : Le système doit être flexible et capable de prendre en charge de nouvelles fonctions et extensions et doit pouvoir fonctionner avec un nouveau composant (RAM, stockage) et avec une modification minimale.

–Extensibilité : Le système doit maintenir ses hautes performances sous pression et ajuster ses paramètres pour répondre à la demande, possibilité d'ajouter des nœuds en cas de montée en charge.

2.4 Product Backlog

Product Backlog est une liste de toutes les tâches connues pour être traitées par le produit. Il s'agit de la seule source des besoins pour toute modification du produit. Les besoins sont précisés par les user stories. Un "user story" se présente sous la forme suivante (voir tableau 2.1) :

id	Theme	Acteur	Description	Priorité
id 1	gérer code source	Développeur	Permet de déposer le code ou le récupérer	élevée
id2	Suivre build d'application	Développeur	Peut voir la compilation de l'application	élevée
id3	analyser qualité de code	Ingénieur DEVOPS	Permet d'analyser le code source d'un projet et de détecter les erreurs de qualité avec SonarQube .	élevée
id 4	gérer liste des images docker	Ingénieur DEVOPS	Consiste à répertorier et organiser les images Docker qui ont été créées .	élevée
id 5	gérer les artefacts d'application	Ingénieur DEVOPS	Consiste à stocker et à gérer les différents composants d'une application, les bibliothèques, les configurations, les scripts et autres ressources .	élevée
id 6	Configurer l'architecture cloud	Ingénieur DEVOPS	Consiste à concevoir et à déployer une infrastructure informatique dans le cloud, en utilisant des services cloud pour répondre aux besoins.	élevée
id 7	Configurer le pipeline	Ingénieur DEVOPS	Est un processus important pour automatiser le processus de développement, de test et de déploiement d'une application.	élevée
id 8	Surveiller l'état du cluster	Ingénieur DEVOPS	Surveille l'état de tous les nœuds dans le cluster pour s'assurer qu'ils sont en bon état de fonctionnement et qu'ils répondent aux demandes des applications.	élevée
id 9	gérer la configuration du déploiement	Ingénieur DEVOPS	Assure que les applications sont déployées de manière cohérente et fiable	élevée
id 10	accès à l'application déployée	Client	s'assure qu'elle est accessible uniquement par les utilisateurs autorisés.	élevée

TABLE 2.1 – Tableau du Product Back-log

2.5 Diagramme de cas d'utilisation Global

Le diagramme de cas d'utilisation global est utilisé pour une représentation du comportement fonctionnel d'un système logiciel.

Le diagramme ci-dessous présente les cas d'utilisation généraux et les acteurs qui interagissent avec le système(Voir figure 2.1).

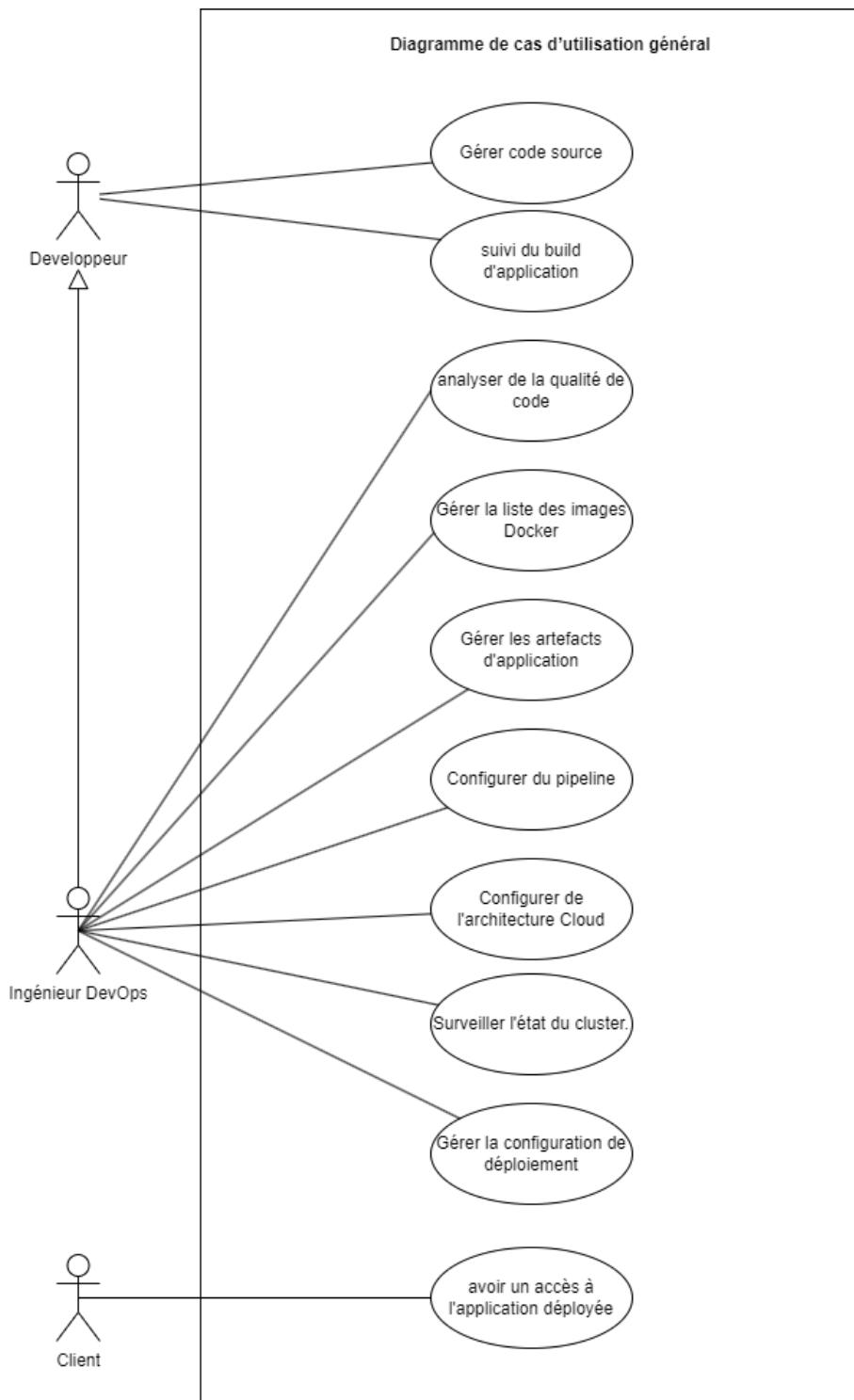


FIGURE 2.1 – Diagramme de cas d'utilisation Globale

Dans ce qui suit , le raffinement des différents cas d'utilisation.

2.5.1 Cas d'utilisation "Gérer code source"

Dans cette section, nous décrirons de façon détaillée ce cas d'utilisation.

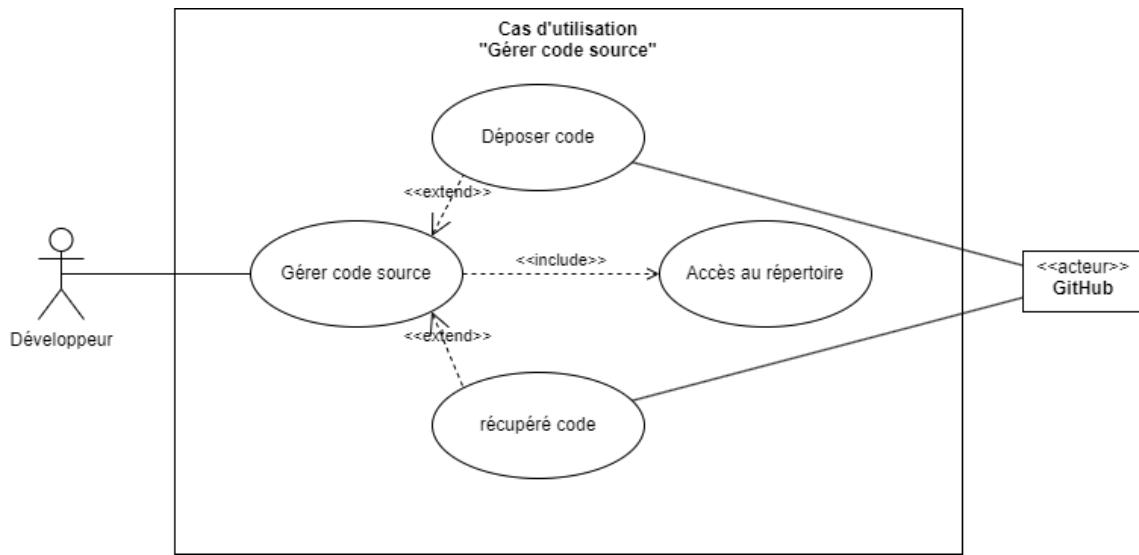


FIGURE 2.2 – Cas d'utilisation :Gérer code source

Pour expliquer le diagramme de cas d'utilisation, nous représentons la description textuelle du principales fonctionnalités mentionnées ci-dessus (Voir tableau 2.2) :

Titre	Gérer code source
Acteur	Développeur
Description	le développeur peut déposer ou récupérer le code à travers le command "pull" ou "push".
Pré conditions	Une connexion établie entre le PC du développeur et le répertoire Git de l'application.
Post conditions	Code envoyé au répertoire git.
Scénario nominal	1- Avoir un accès au répertoire. 2 - Gérer code source. 3 - Déposer ou récupérer le code dans le répertoire github.
Scénario alternatif	1- La connexion entre la machine du développeur et Git ne peut pas être établie. 2 - Le code ne peut pas être envoyé. 3 - Retour à l'étape 1 du Scénario nominal.

TABLE 2.2 – Description de cas d'utilisation :Gérer code source

2.5.2 Cas d'utilisation "Suivi de la build d'application"

Dans cette section, nous décrirons de façon détaillée ce cas d'utilisation.

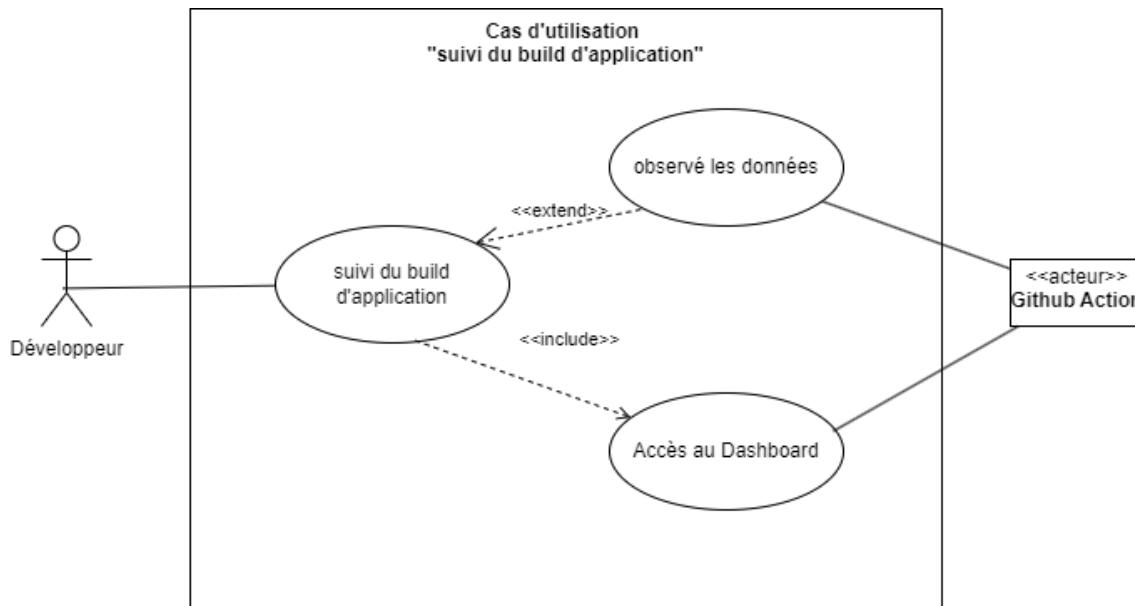


FIGURE 2.3 – Cas d'utilisation :Suivi la build d'application

Pour expliquer le diagramme de cas d'utilisation, nous représentons la description textuelle des principales fonctionnalités mentionnées ci-dessus(voir tableau 2.3) :

Titre	Suivi du build de l'application
Acteur	Développeur
Description	Le développeur peut suivre la construction d'application pour connaître si la modification ajoutée au code source est valide ou non.
Pré conditions	Une connexion établie entre le développeur et Github actions.
Post conditions	Affichage de Dashboard Github Actions.
Scénario nominal	1- Avoir un accès au Dashbord 2- Suivi du build de l'application 3- L'observation de la construction de l'application rapporte un succès.
Scénario alternatif	1 - La connexion entre développeur et Github Actions ne peut pas être établie. 2 - Retour à l'étape 1 du Scénario nominal.

TABLE 2.3 – Description de cas d'utilisation :Suivi du build de l'application

2.5.3 Cas d'utilisation "Gérer la configuration de déploiement"

Dans cette section, nous décrirons de façon détaillée ce cas d'utilisation (voir figure 2.4).

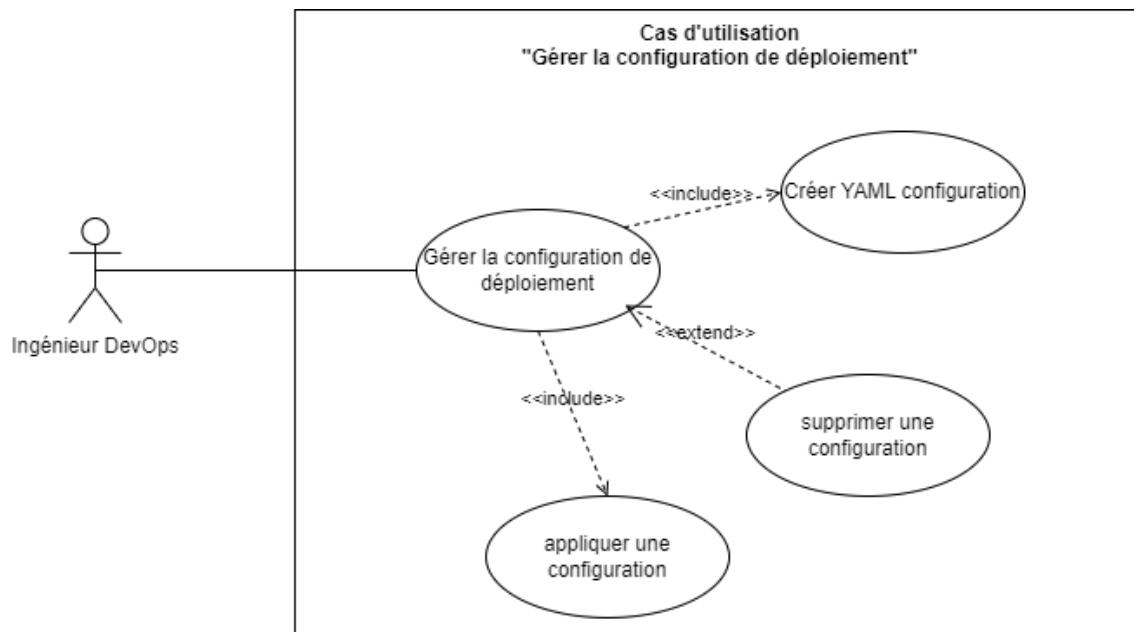


FIGURE 2.4 – Cas d'utilisation :Gérer la configuration de déploiement

Pour expliquer le diagramme de cas d'utilisation, nous représentons la description textuelle des principales fonctionnalités mentionnées ci-dessus (voir table 2.4) :

Titre	Gérer la configuration de déploiement
Acteur	Ingénieur DevOps
Description	L'ingénieur DevOps gère la configuration de déploiement pour assurer la meilleure performance du cluster.
Pré conditions	Fonctionnement correct du cluster EKS.
Post conditions	Une configuration ou modification d'une configuration déjà existante.
Scénario nominal	1 - Lancer la configuration du déploiement. 2 - La configuration est lancée dans le contrôle plane d'EKS.
Scénario alternatif	1 - La nouvelle configuration n'est pas flexible. 2 - Un problème apparaît causé par la nouvelle configuration. 3 - Retour à l'étape 1 du Scénario nominal.

TABLE 2.4 – Description de cas d'utilisation :Gérer la configuration de déploiement

2.5.3.1 Cas utilisation "Gérer liste des images Docker"

Dans cette section, nous décrirons de façon détaillée le cas d'utilisation (voir figure 2.5).

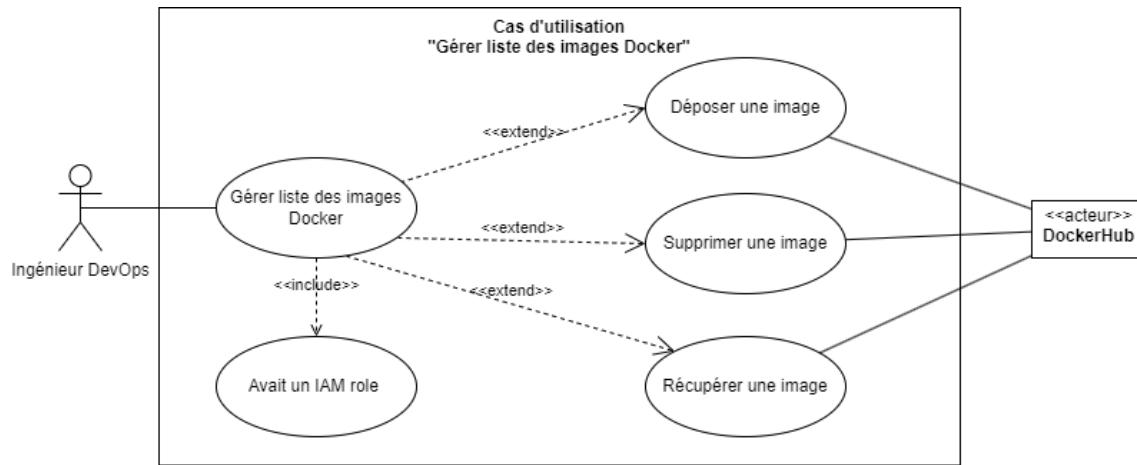


FIGURE 2.5 – Cas d'utilisation :Gérer liste des images Docker

Pour expliquer le diagramme de cas d'utilisation, nous représentons la description textuelle des principales fonctionnalités mentionnées ci-dessus (voir table 2.5) :

Titre	Gérer liste des images Docker
Acteur	Ingénieur DevOps
Description	L'ingénieur DevOps gère la liste des images Docker
Pré conditions	Connexion au répertoire de l'application dans ECR.
Post conditions	Une image modifiée ,supprimée ou ajoutée.
Scénario nominal	1 - Se connecter ou l'enregistrement de l'image n'est pas valide. 2 - Configurer liste des images Docker . 2 - Déposer ou récupérer le code dans le répertoire ECR. 3 - L'image est enregistrée dans la répertoire ECR.
Scénario alternatif	1 - La connexion ou l'enregistrement de image n'est pas valide. 2 - Echec du dépôt ou récupération l'image Docker . 3 - Retour à l'étape 1 du Scénario nominal.

TABLE 2.5 – Description de cas d'utilisation :Gérer liste des images Docker

2.5.4 Cas utilisation "Gérer les artefacts d'application"

Dans cette section, nous décrirons de façon détaillée ce cas d'utilisation (voir figure 2.6).

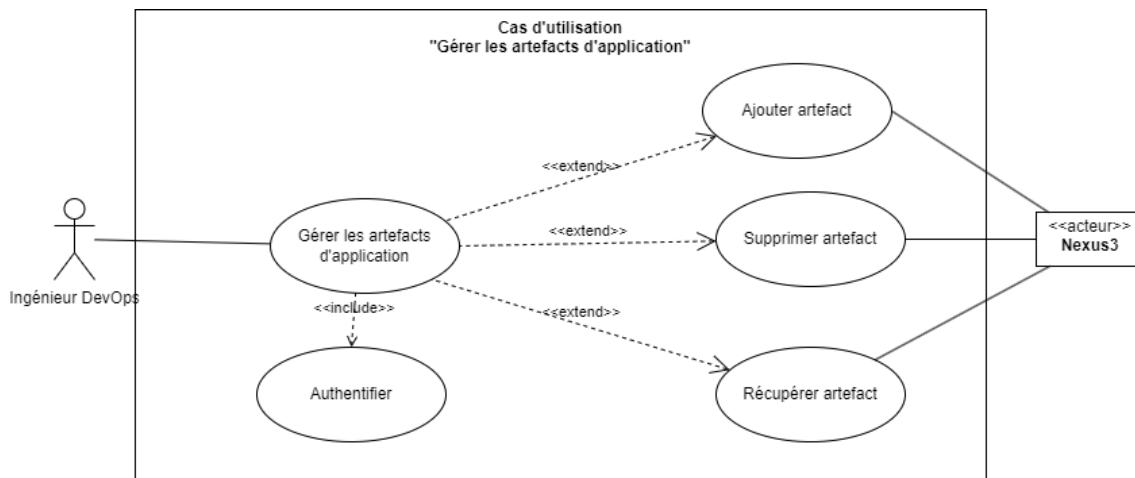


FIGURE 2.6 – Cas d'utilisation :Gérer les artefacts d'application

Pour expliquer le diagramme de cas d'utilisation, nous représentons la description textuelle des principales fonctionnalités mentionnées ci-dessus (voir table 2.6) :

Titre	Gérer les artefacts de l'application
Acteur	Ingénieur DevOps
Description	L'ingénieur DevOps gère la liste des artefacts pour assurer l'enregistrement de différentes versions d'application.
Pré conditions	Connexion HTTP au serveur Nexus.
Post conditions	Une répertoire avec les différentes artefacts de l'application.
Scénario nominal	1 - Saisir les données d'authefication pour accéder à l'interface . 2 - Configurer les artifacts d'application . 3 - la connexion est réussite et la modification dans le répertoire est enregistrée dans Nexus .
Scénario alternatif	1 - La connexion échoue ou la modification n'est pas enregistrée. 2 - Retour à l'étape 1 du Scénario nominal.

TABLE 2.6 – Description de cas d'utilisation :Gérer les artefacts d'application

2.6 Conception du système

Dans ce qui suit, nous allons présenter les différents diagrammes de conception.

2.6.1 Diagramme de séquence global

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique[?](voir figure 3.1).

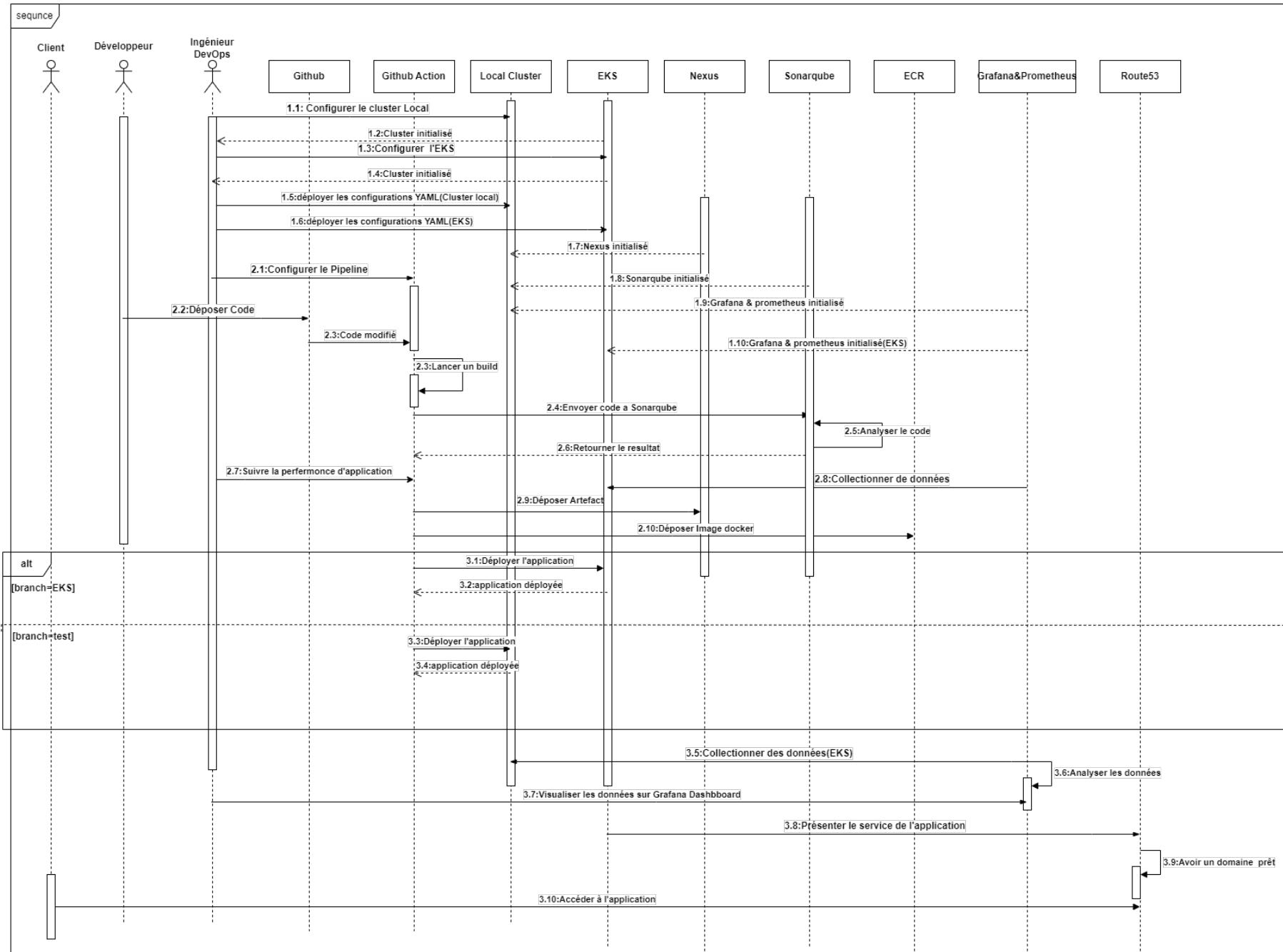


FIGURE 2.7 – Diagramme de séquence Global

2.6.2 Diagramme de séquence Détailé

Pour une meilleure présentation et illustration de projet nous allons décrire les différents cas d'utilisations principaux avec leurs diagrammes de séquence détaillés.

2.6.2.1 Diagramme de séquence «Configuration de l'architecture cloud»

Nous détaillons ci-dessous les interactions du diagramme de séquence « Configuration de l'architecture cloud »

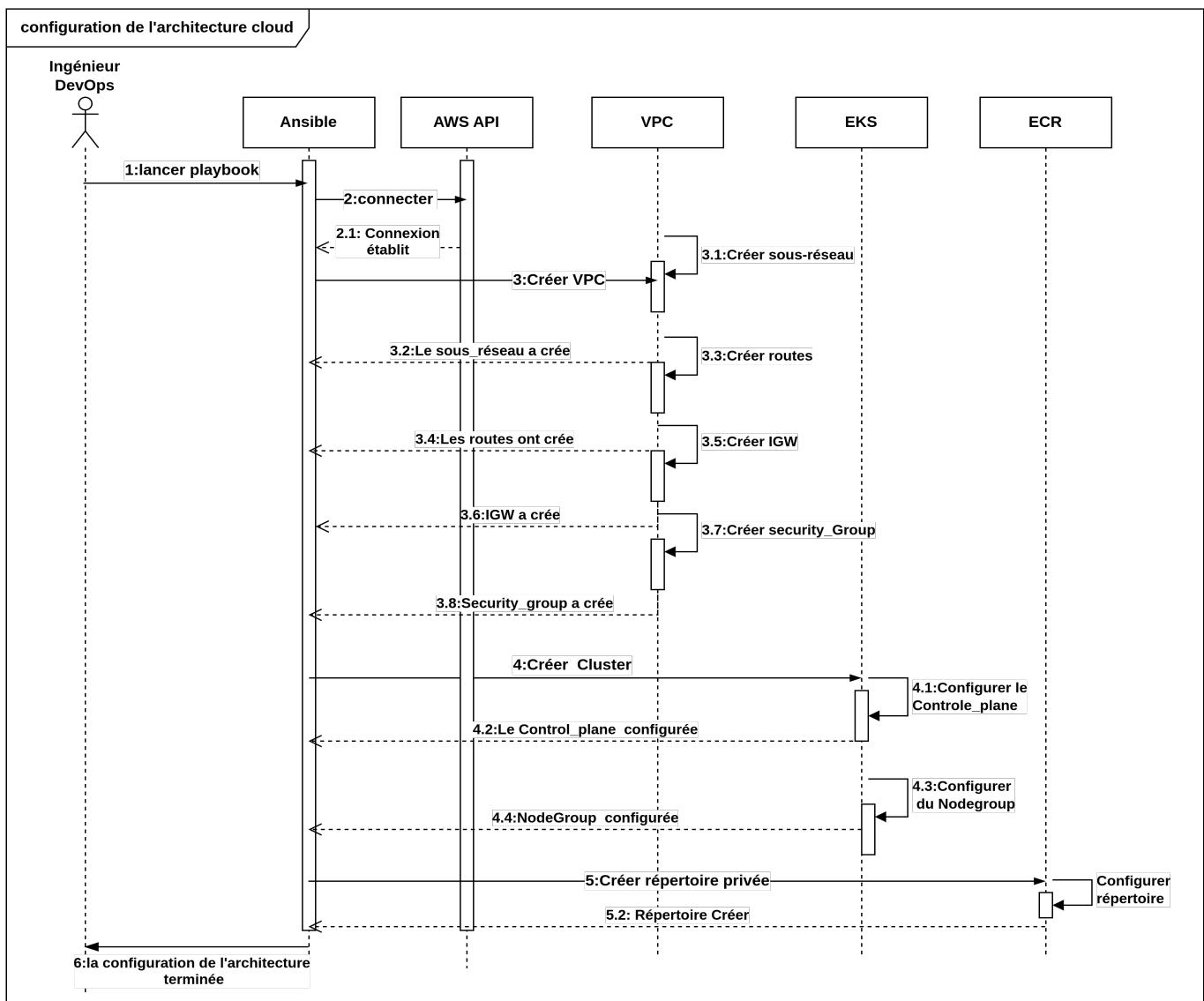


FIGURE 2.8 – Diagramme de séquence « Configuration de l'architecture cloud »

2.6.2.2 Diagramme de séquence « Configuration de cluster local »

Nous détaillons ci-dessous les interactions de diagramme de séquence « Configuration de cluster local » (voir figure 3.3).

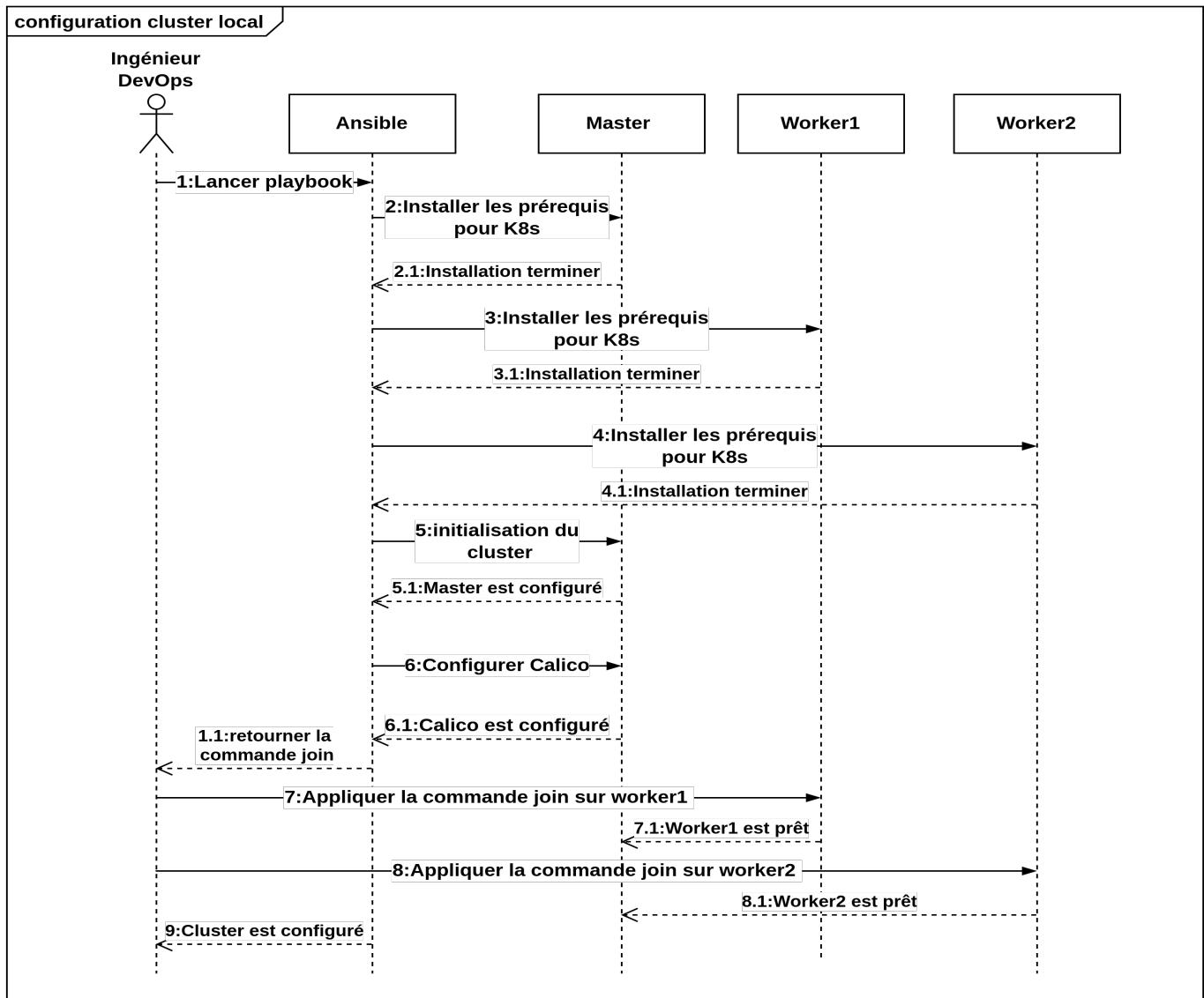


FIGURE 2.9 – Diagramme de séquence « Configuration de cluster local »

2.6.2.3 Diagramme de séquence « Configuration du pipeline »

Nous détaillons ci-dessous les interactions de diagramme de séquence « Configuration du pipeline » (voir figure 3.4) .

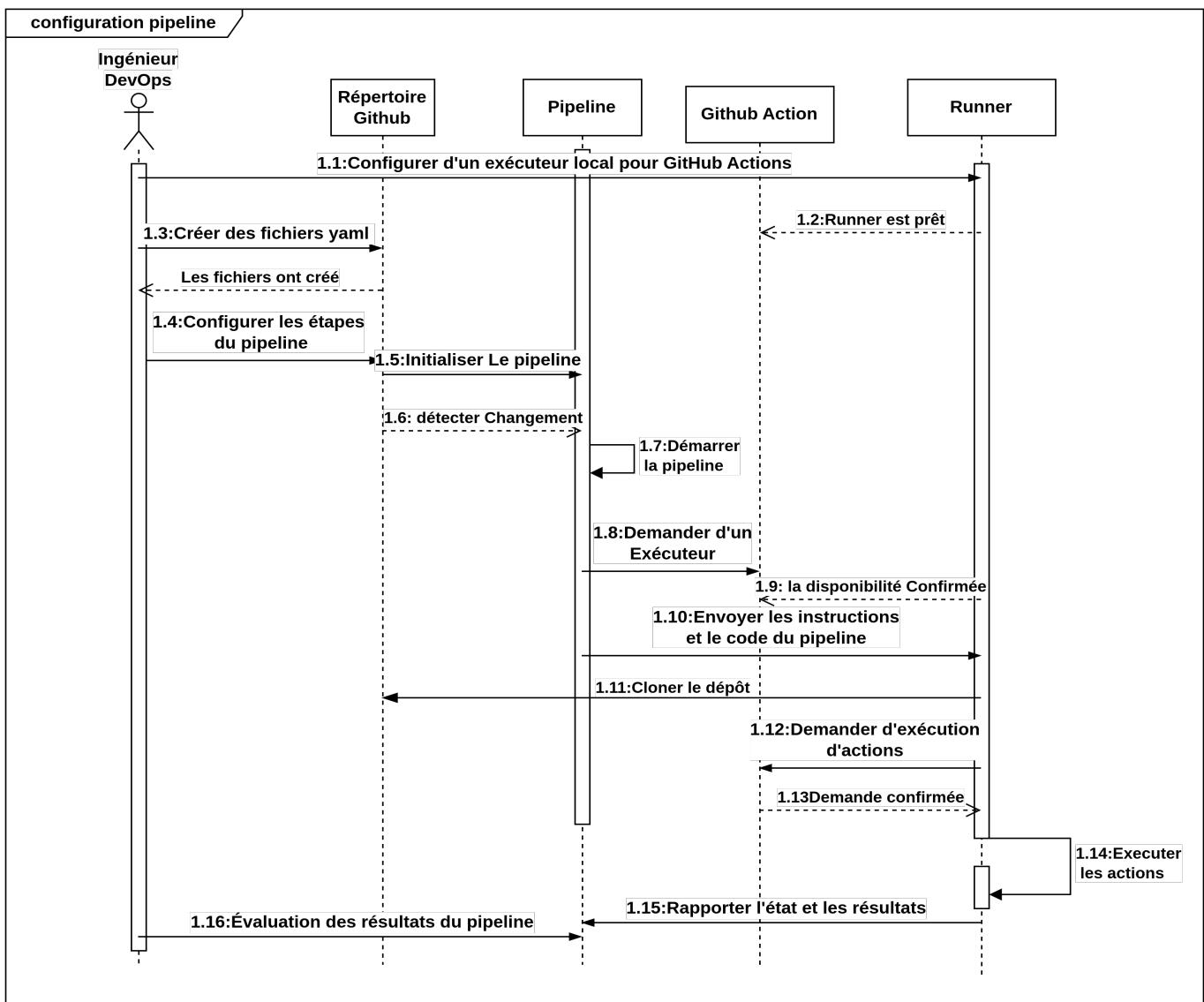


FIGURE 2.10 – Diagramme de séquence « Configuration du pipeline »

2.6.3 Diagramme de classe

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces du système ainsi que leurs relations. La figure ci-dessous représente le diagramme de classe de notre travail(voir figure 4.5)[?].

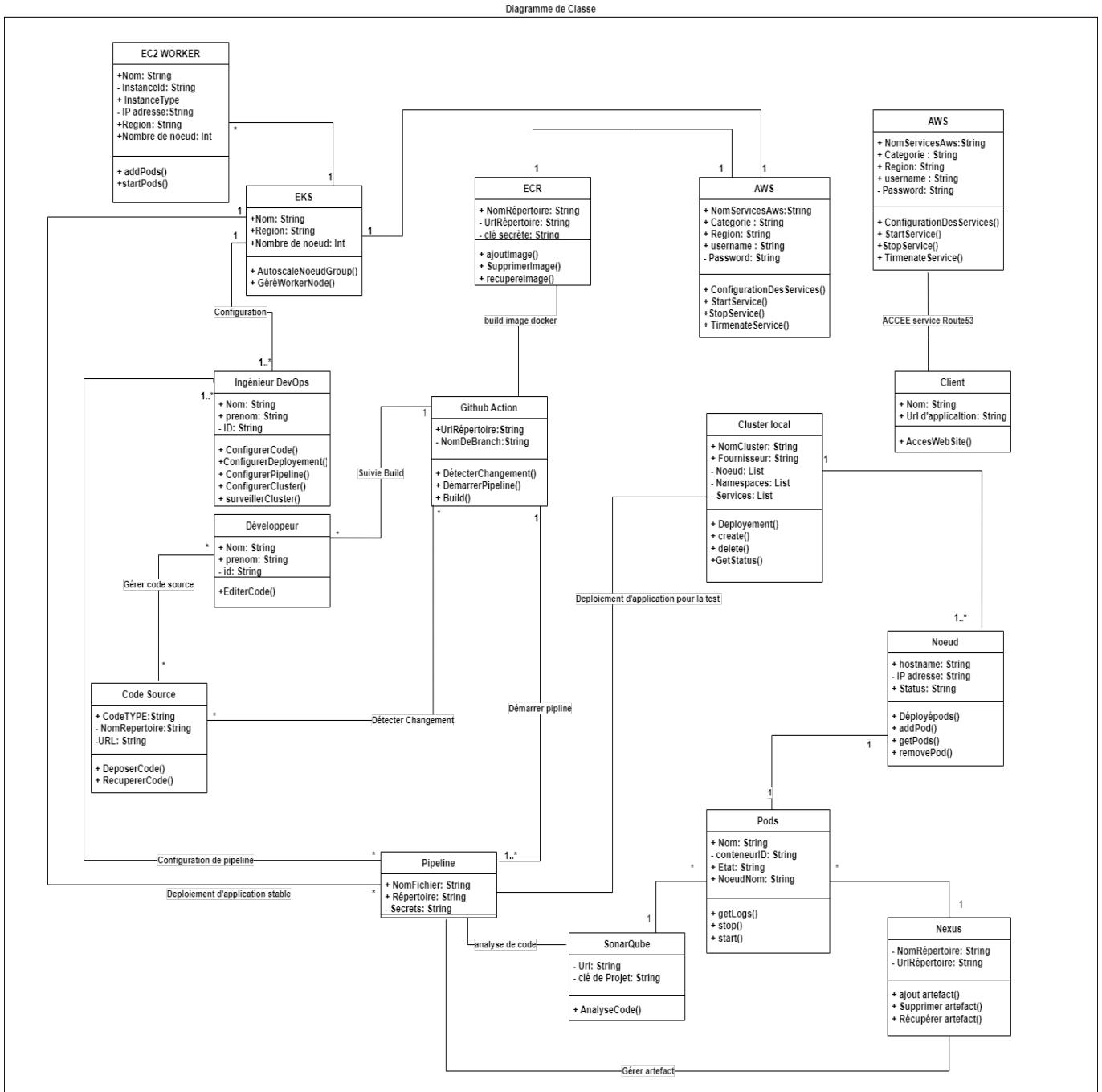


FIGURE 2.11 – Diagramme de classe global

2.6.4 Diagramme de déploiement

Un diagramme de déploiement est une vue statique qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les composants du système sont répartis ainsi que leurs relations entre eux (voir figure 4.6)[?].

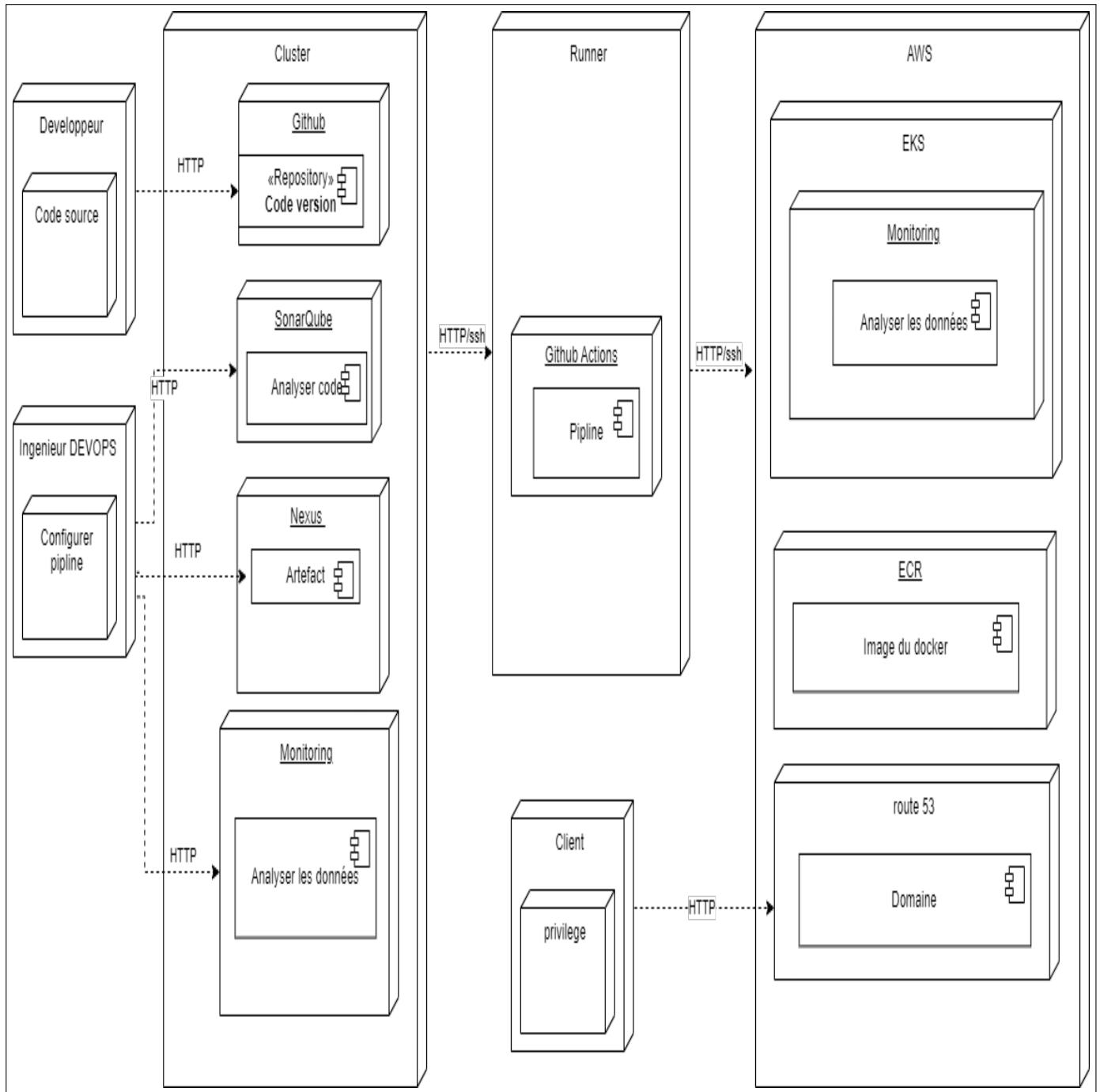


FIGURE 2.12 – Diagramme de déploiement global

2.7 Architecture technologique de la solution

Dans cette partie nous allons présenter l'architecture globale de notre projet(voir figure 3.8).

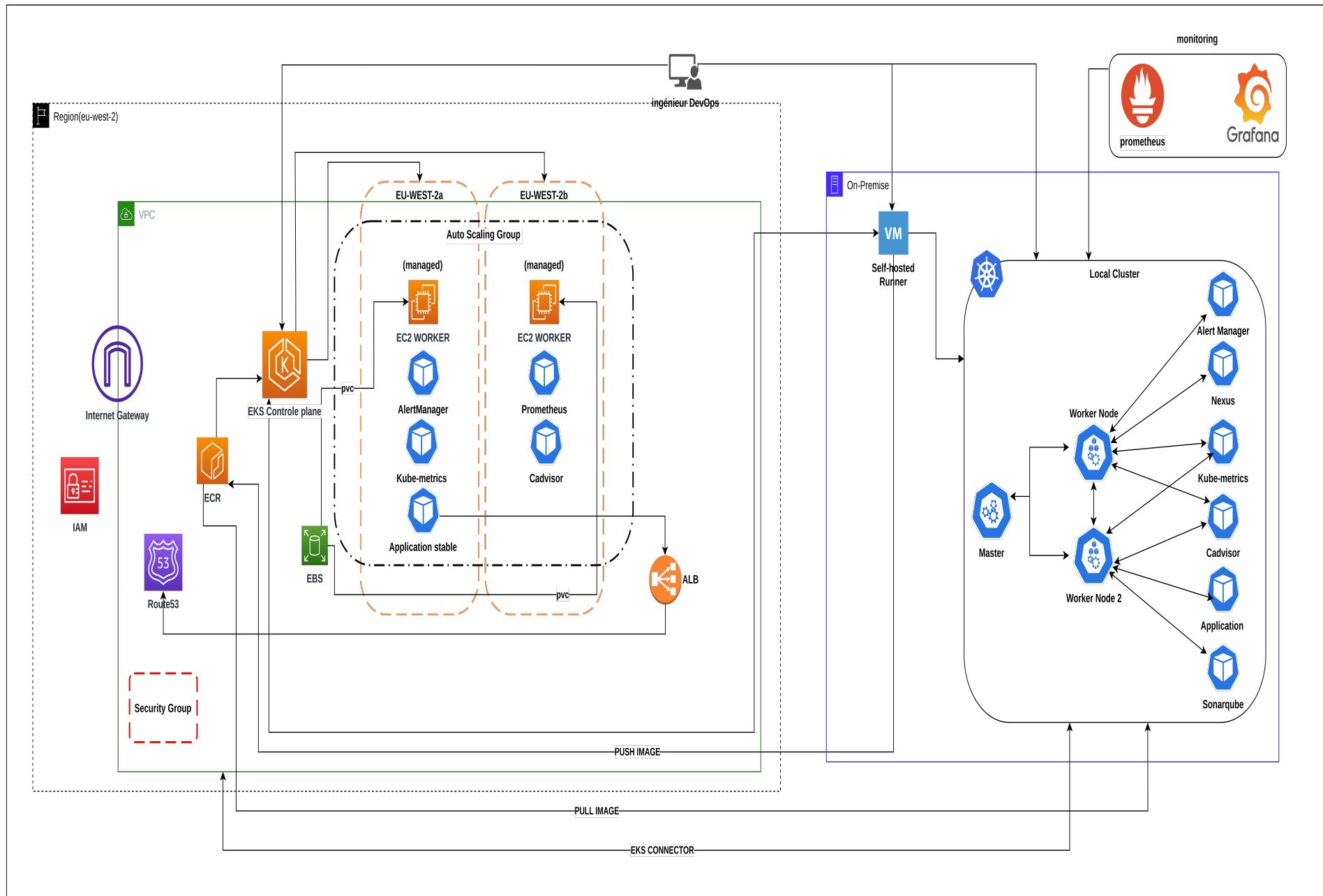


FIGURE 2.13 – Architecture global

Pour plus de clarté, nous allons diviser notre architecture globale en deux parties.

–Partie local :

La figure (figure : 3.9) représente les ressources utiliser localement dans le projet.

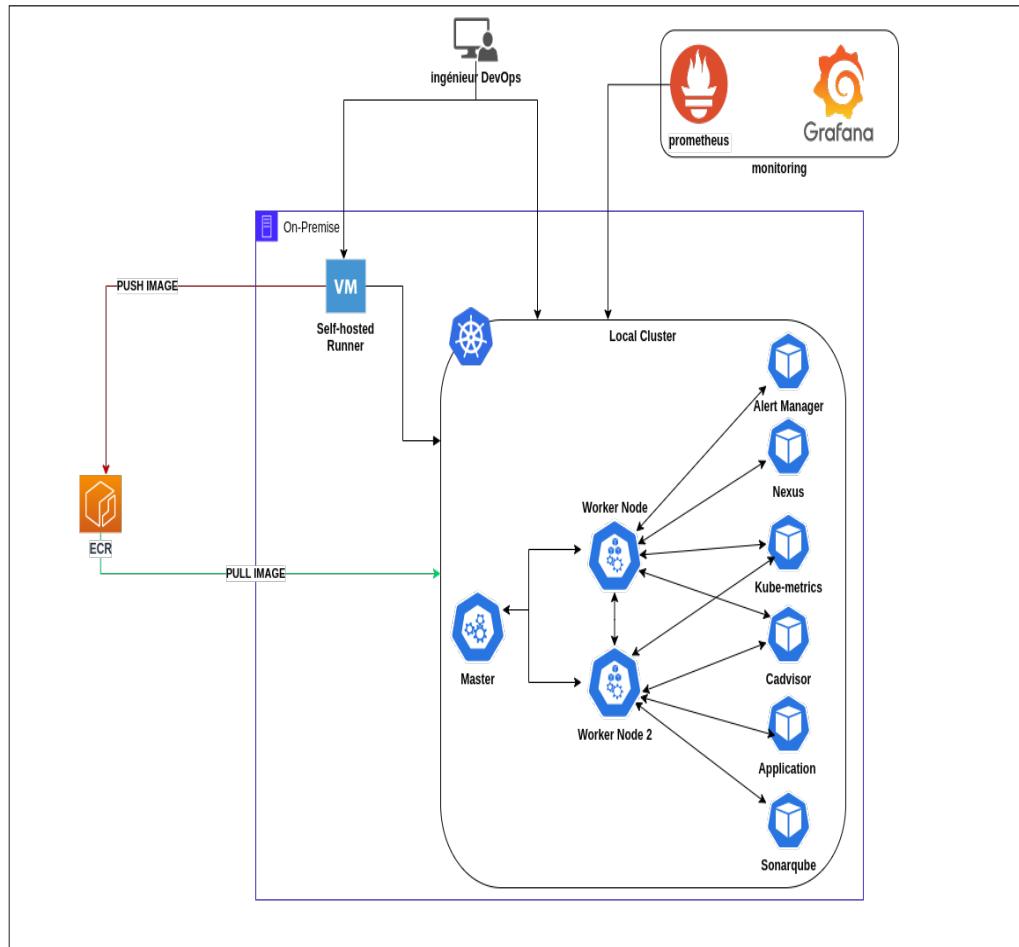


FIGURE 2.14 – Partie local

- Le cluster local dans l'architecture globale fonctionne comme un environnement dédié pour les tests et le développement. Il est configuré et géré par l'ingénieur devops, qui dispose des droits et des permissions nécessaires. Le cluster local est composé de plusieurs composants clés qui facilitent le développement efficace et tester les flux de travail.

- Dans le cluster local, plusieurs nœuds ou machines virtuelles sont fournis pour créer un environnement similaire à la configuration de production. Ces nœuds exécutent des conteneurs ou d'autres outils de test, permettant possible de simuler le comportement et les performances de l'environnement de production localement. L'ingénieur de devons supervise la configuration et la gestion du cluster local, en assurant sa disponibilité et sa stabilité pour les activités de test et de développement.

- Pour surveiller le cluster local, l'ingénieur devops déploie Grafana et Prometheus, qui fournissent des informations précieuses sur les performances et la santé du cluster. Ces outils de surveillance permettent de suivre l'utilisation des

ressources, de détecter et de résoudre les problèmes rapidement, ce qui permet l'amélioration continue et l'optimisation de l'environnement de test local.

- L'ingénieur devops configure également un exécuteur local pour GitHub Actions. Cet exécuteur permet de déclencher des pipelines. Dans le cas de la branche principale, un pipeline démarre et les actions de déploiement nécessaires sont exécutées sur le cluster local. Cela permet l'ingénieur devops de valider les changements dans un environnement qui ressemble beaucoup à la configuration de production, assurant robustesse et fiabilité.

- L'ingénieur de devops configure l'environnement pour extraire les images de conteneurs d'un dépôt ECR privé, qui stocker et gérer les images de conteneurs. Les images du dépôt sont séparées par des versions telles que 1.0.0 , 0.0.1 pour les tests.

–Partie AWS :

La figure ci-dessous représente les ressources cloud utiliser dans notre projet.

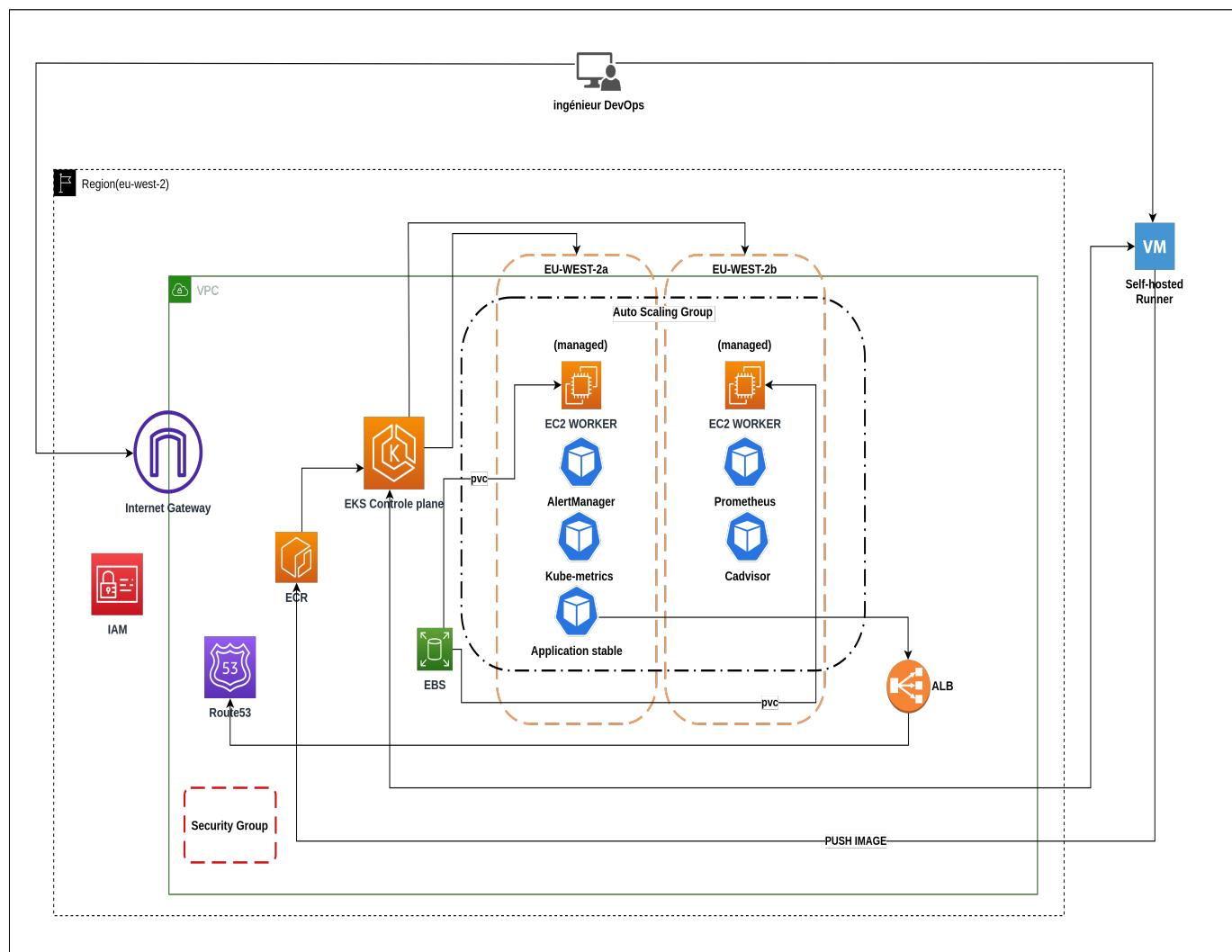


FIGURE 2.15 – Partie AWS

- Le cluster EKS se compose de noeuds de travail, qui sont des instances EC2 qui exécutent les conteneurs contenant la charge de travail de l'application. L'in-

génieur de devops assure l'approvisionnement et la mise à l'échelle appropriés des nœuds de travailleurs en fonction des exigences de l'application et de la charge de travail. Cela implique de configurer des politiques de mise à l'échelle automatique pour ajuster automatiquement le nombre de nœuds de travail afin de gérer efficacement les différents charges de travail.

- Pour gérer le trafic entrant et le distribuer à travers les conteneurs du cluster EKS, un Load balancer est lancé pour aider à atteindre une haute disponibilité et une tolérance aux pannes en transférer les demandes vers les conteneurs, assurant ainsi une performance et une fiabilité optimales de l'application.

- Lorsque la branche EKS est déclenchée par des développeurs leur code source, un pipeline dédié est lancé. Pour exécuter les actions de déploiement nécessaires sur le cluster EKS, un exécuteur configuré par l'ingénieur devops utilise son contexte et ses autorisations pour interagir efficacement avec le cluster. Cela permet un déploiement continu de l'application dans l'environnement de production hébergé sur le cluster EKS.

- Le surveillance du cluster EKS est également une responsabilité de l'ingénieur de devops. Prometheus installer sur EKS, fournissent des informations complètes sur les performances, l'utilisation des ressources et les paramètres de santé du cluster. Ce outil permettant à l'ingénieur de détecter et de résoudre rapidement les problèmes potentiels, assurant le bon fonctionnement de l'environnement de production.

Conclusion

Dans ce chapitre nous avons décrit les différents besoins fonctionnels et non fonctionnels. Aussi, nous avons décrit les différents acteurs de système. Puis nous avons présenté les "User stories" en forme d'un product backlog. Par la suite, nous avons illustrée les cas d'utilisation avec des diagrammes. Puis, nous avons décrit les différents diagrammes de conception. Aussi, nous avons décrit la méthodologie de conception utiliser dans notre projet. Enfin, nous aussi présenter l'architecture global qui expliquer la fonctionnement du projet. Dans le chapitre suivante nous passerons à la phase de réalisation du projet.

Chapitre 3

Réalisation

Introduction

Au cours de ce dernier chapitre, on va exposer le travail accompli, explorer les différent étapes pour réaliser le travail final en détaillons chaque étape et en incluant quelques captures d'écran expliquant le processus de déploiement de la solution.

3.1 Environnement de travail

Dans cette partie, nous présentons les environnements matériels et logiciels utilisés dans le cadre de notre projet.

3.1.1 Environnement matériel

Au cours de les différentes étape de notre projet, nous avons disposé de deux PC ayant les caractéristiques figurantes dans le tableau ci-dessous :

Marque	lenovo ideapad 3	Lenovo Legion slim 7
Processeur	AMD Rayzen 5 3.3GHZ	AMD Rayzen 9 4.10GHZ
Mémoire Vive	16GB RAM	16GB RAM
Stockage	500GB	1To
Système d'exploitation	Fedora 37	Windows 11

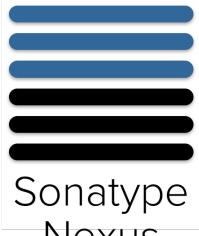
TABLE 3.1 – Caractéristiques des ordinateurs

3.1.2 Environnement Logiciel

Dans cette partie, nous présentons les environnements logiciels différents utilisés.(voir tableau 5.2)

3.1.2.1 Environnement et technologie de développement

Nom	Description	Utilisation
 Visual Studio Code	Visual Studio Code est un éditeur de code source développé par Microsoft.	Nous avons utilisé Visual studio code pour la développement de code et interagi avec github.
	VMware Workstation est un outil de virtualisation de poste de travail créé par la société VMware, il peut être utilisé pour mettre en place un environnement de test pour développer de nouveaux logiciels, ou pour tester l'architecture complexe d'un système d'exploitation avant de l'installer réellement sur une machine physique.[?]	Nous avons utilisé Vmware pour créer diffèrent virtual machine pour deployée le cluster.
	Latex est un logiciel de préparation de documents qui utilise le programme de composition TeX pour formater sa sortie, et est lui-même écrit dans le langage de macro TeX. Le latex est largement utilisé dans les universités pour la publication de documents scientifiques dans de nombreux domaines, y compris les mathématiques, l'informatique, etc.	Nous avons utilisé Latex pour rédaction de notre rapport.
	Github est un site développé par Chris Wanstrath, PJ Hyett et Tom Preston-Werner. Fournir divers services qui aident les développeurs à gérer leur code source à l'aide du logiciel de gestion de versions Git.	Nous avons utilisé Github pour déposer notre code.
	Docker est une plate-forme qui rend l'exécution, le test et la livraison de logiciels plus rapide avec sa virtualisation au niveau OS qui permet de séparer l'application dans des conteneurs.	Nous avons utilisé docker dans la création des images docker.

 GitHub Actions	<p>GitHub Actions est une plateforme CI/CD (Continuous Integration and Continuous Delivery) qui vous permet d'automatiser votre pipeline de construction, de test et de déploiement. Vous pouvez créer des flux de travail qui construisent et testent chaque requête merge ,pull ou push sur votre dépôt.</p>	<p>Nous avons utilisé Github Action pour la création et l'exécution de notre pipeline.</p>
 Sonatype Nexus	<p>Nexus est gestionnaire de dépôt qui vous permet de proxy, collecter et gérer différents artefacts et image docker et rendre facile de distribuer votre logiciel.</p>	<p>Nous avons utilisé Nexus pour gérer notre artefact de l'application développer.</p>
	<p>sonarqube est une plate-forme open source développée par SonarSource pour le contrôle de code qualité et les rapports de code statique, elle supporte différents langages et frameworks de programmation.</p>	<p>Nous avons utilisé Sonarqube pour l'analyse de notre code source et vérifier la présence des défauts.</p>
 ANSIBLE	<p>Ansible est un outil d'automatisation qui fournit une infrastructure en tant que code créé par Michael DeHaan et acquit par Red Hat, il fournit des outils pour la configuration, le déploiement d'application et le provisioningnement.</p>	<p>Nous avons utilisé Ansible pour automatiser la configuration de cluster local et les ressources cloud.</p>
 kubernetes	<p>Kubernetes est un système open source qui automatiser le déploiement, la mise à l'échelle et la gestion des applications dans des conteneurs</p>	<p>Nous avons utilisé Kubernetes pour la déploiement des différent application et outils.</p>
 Prometheus	<p>Prometheus est un système open source développé par soundcloud pour la collection de données et de métriques à l'aide d'un modèle http pull.</p>	<p>Nous avons utilisé Prometheus pour la collection des différent données des outils,clusters et applications.</p>
 Grafana	<p>Grafana est une application Web d'analyse open source et visualisation interactive. Il fournit des graphiques et des alertes lorsqu'il est connecté à des sources de données prises en charge.</p>	<p>Nous avons utilisé Grafana pour la création des alerts et visualisation des différent données collecter par prometheus en forme des différent diagrammes.</p>

	Slack est un logiciel de messagerie développé par Slack Technologies. Slack est développé pour les communications professionnelles et organisationnelles.	Nous avons utilisé Slack pour recevoir les alertes créées par Grafana.
	EKS is a service to create a managed Kubernetes cluster by Amazon. It offers different add-ons that help to interact with other services like EBS, ALB or NLB.	Nous avons utilisé EKS pour créer un cluster pour l'environnement de production.
	Amazon elastic container Registry (Amazon ECR) est un service qui registre les images Docker dans des répertoires soit public soit privé.	Nous avons utilisé ECR pour enregistrer notre différentes images Docker.
	EC2 (Elastic Compute Cloud) est un service fourni par Amazon qui facilite la location d'ordinateurs virtuels pour exécuter leurs propres applications.	Nous avons utilisé EC2 pour créer des nœuds pour le cluster EKS.

TABLE 3.2 – Environnement et technologies de développement

3.1.3 Langage Informatique

Le tableau ci-dessous représente les différents langages utilisés dans le projet.

Nom	Description	Utilisation
	React.js est une bibliothèque open source utilisée pour construire des interfaces, il peut également être utilisé sur iOS, Android et web.	Cette langue est utilisée dans l'application que nous avons testée dans notre projet.
	YAML (Ain't Markup Language) un langage de sérialisation de données développé par Clark Evans, Ingy döt Net et Oren Ben-Kiki, utilisé pour créer des fichiers de configuration.	Nous avons utilisé ce langage pour créer différents playbooks.

TABLE 3.3 – Langage informatique

3.1.4 Environnement de Conception

Le tableau ci-dessous représente les différents outil de conception utilisé dans le projet.

Nom	Description	Utilisation
 draw.io	draw.io est une application de source libre qu'il est utiliser pour illustrer des diagrammes.	Ce outil est utilisée pour les différents diagrammes dans notre projet.
	Canva est un outil de conception graphique en ligne gratuit,utiliser pour créer des présentations, des logos et plus encore	Nous avons utilisé canva pour désigner quelque diagrammes.

TABLE 3.4 – Environnement de conception

3.2 Sprint1 :Configuration de l'architecture cloud

Pour ce partie nous allons configuré tout les ressources cloud que nous besoin pour la projet.

3.2.1 Configuration des ressources cloud

Pour la configuration des ressources cloud nous avons préparer un playbook ansible qui créer les ressources nécessaire dans AWS comme VPC,Subnets,route,security groups,EKS and ECR.ou début en definit les différent variables pour les ressources

```

1  ---
2  - name: Create AWS RESSOURCES
3  hosts: localhost
4  gather_facts: yes
5  vars:
6      cluster_name: EKS-CLUSTER
7      worker_node_instance_type: t3.medium
8      worker_name: worker-group-1
9      vpc_name: EKS
10     region: eu-west-2
11     vpc_cidr_block: 172.30.0.0/16
12     subnets:
13         - name: public_subnet1
14             cidr_block: "172.30.10.0/24"
15             az: "eu-west-2a"
16             map_public: true
17         - name: public_subnet2
18             cidr_block: "172.30.20.0/24"
19             az: "eu-west-2b"
20             map_public: true
21         - name: private_subnet
22             cidr_block: "172.30.100.0/24"
23             az: "eu-west-2c"
24             map_public: false
25     my_ip: "{{ lookup('pipe', 'curl -sS ifconfig.me') }}/32"
26 tasks:

```

FIGURE 3.1 – Les variables de playbook ansible

Après la définition des variable dans le playbook on passe a les tâches qui sera exécuter par Ansible :

- Tâche « Créer VPC » :

```

28     - name: Create VPC
29         ec2_vpc_net:
30             name: "{{ vpc_name }}"
31             cidr_block: "{{ vpc_cidr_block }}"
32             region: "{{ region }}"
33             state: present
34             register: vpc
35     - name: Set VPC fact
36         set_fact:
37             vpc_id: "{{ vpc.vpc.id }}"

```

FIGURE 3.2 – Tâche de VPC

- Tâche « Créer les sous-réseaux » :

```

40   - name: Create Subnets
41     ec2_vpc_subnet:
42       vpc_id: "{{ vpc_id }}"
43       cidr: "{{ item.cidr_block }}"
44       state: present
45       region: "{{ region }}"
46       az: "{{ item.az }}"
47       map_public: "{{ item.map_public }}"
48       with_items: "{{ subnets }}"
49       register: subnets
50   - name: Set subnet facts for public subnets
51     set_fact:
52       "{{ item.item.name }}_id": "{{ item.subnet.id }}"
53       with_items: "{{ subnets.results }}"
54       when: item.item.map_public == true
55   - name: Set subnet facts
56     set_fact:
57       subnet_id: "{{ item.subnet.id }}"
58       with_items: "{{ subnets.results }}"
59

```

FIGURE 3.3 – Tâche des sous-réseaux

- Tâche « Créer passerelle Internet » :

```

62   - name: Create Internet Gateway
63     ec2_vpc_igw:
64       vpc_id: "{{ vpc_id }}"
65       state: present
66       region: "{{ region }}"
67       resource_tags:
68         Name: "{{ vpc_name }}-IGW"
69       register: internet_gateway
70   - name: Set Internet Gateway fact
71     set_fact:
72       internet_gateway_id: "{{ internet_gateway.gateway_id }}"

```

FIGURE 3.4 – Tâche de passerelle internet

- Tâche « Créer route » :

```

76   - name: Gather information about any VPC route table within VPC with ID
77     amazon.aws.ec2_vpc_route_table_info:
78       filters:
79         vpc_id: "{{ vpc_id }}"
80       register: existing_route_tables
81   - name: Create Default Route Table
82     ec2_vpc_route_table:
83       vpc_id: "{{ vpc_id }}"
84       region: "{{ region }}"
85       subnets:
86         - "{{ public_subnet2_id }}"
87         - "{{ public_subnet1_id }}"
88       state: present
89       routes:
90         - dest: 0.0.0.0/0
91           gateway_id: "{{ internet_gateway_id }}"
92       register: routes
93       when: existing_route_tables.route_tables | length == 0
94

```

FIGURE 3.5 – Tâche de route internet

- Tâche « Créer la sécurité groupe » :

```

50
97     - name: Create Security Group
98       ec2_group:
99         name: "{{ vpc_name }}-SecurityGroup"
100        description: "Security Group for {{ vpc_name }} VPC"
101        vpc_id: "{{ vpc_id }}"
102        region: "{{ region }}"
103        rules:
104          - proto: tcp
105            from_port: 22
106            to_port: 22
107            cidr_ip: "{{ my_ip }}"
108          - proto: all
109            from_port: 0
110            to_port: 0
111            cidr_ip: "192.168.1.0/24"
112        register: security_group
113    - name: Set Security Group fact
114      set_fact:
115        security_group_id: "{{ security_group.group_id }}"

```

FIGURE 3.6 – Tâche de sécurité groupe.

- Tâche « Créer EKS cluster » :

```

118
119     - name: Create an Amazon EKS Cluster
120       community.aws.eks_cluster:
121         name: EKS-CLUSTER
122         version: '1.26'
123         role_arn: arn:aws:iam::789187835508:role/eksRole
124         subnets:
125           - "{{ public_subnet2_id }}"
126           - "{{ public_subnet1_id }}"
127         security_groups:
128           - "{{ security_group.group_id }}"
129         wait: true
130         region: eu-west-2
131       register: eks_cluster
132

```

FIGURE 3.7 – Tâche de cluster EKS.

- Tâche « Créer Noeud de travail pour le cluster EKS » :

```

134
135     - name: Create a worker node group for the Amazon EKS cluster
136       community.aws.eks_nodegroup:
137         cluster_name: "{{ cluster_name }}"
138         name: worker-group-1
139         wait: yes
140         node_role: arn:aws:iam::789187835508:role/eksworkers
141         instance_types:
142           - t2.micro
143         scaling_config:
144           min_size: 4
145           desired_size: 4
146           max_size: 5
147         subnets:
148           - "{{ public_subnet2_id }}"
149           - "{{ public_subnet1_id }}"
150       register: node_group
151

```

FIGURE 3.8 – Tâche de noeud de travail.

- Tâche « Créer répertoire ECR » :

```

172      - name: Create ECR Private Repo
173        community.aws.ecs_ecr:
174          name: horizonweb
175          state: present
176          image_tag_mutability: immutable
177          register: ECR

```

FIGURE 3.9 – Tâche de ECR

- Résultat de l'exécution de playbook (voir figure 5.10) :

```

changed: [localhost]

TASK [Update kubeconfig] ****
**
changed: [localhost]

TASK [Install EBS CSI Driver addon] ****
**
changed: [localhost]

TASK [Create ECR Private Repo] ****
**
changed: [localhost]

TASK [Print VPC Details] ****
**
ok: [localhost] => {
    "msg": "VPC ID: vpc-0ffedd2f39ddf31e2\nSubnets ID: subnet-0535f6b6886332791\nInternet Gateway ID: igw-0d1210c6c5fd9372f\nSecurity Group ID: sg-0f3651dea9b67f787"
}

TASK [Output Cluster Information] ****
**
ok: [localhost] => {
    "msg": "EKS Cluster: EKS-CLUSTER\nWorker Node Instance Type: t2.micro\nWorker Node Name: worker-group-1"
}

TASK [ECR Information] ****
**
ok: [localhost] => {
    "msg": "ECR: {'state': 'present', 'created': True, 'changed': True, 'repository': {'repositoryArn': 'arn:aws:ecr:eu-west-2:789187835508:repository/horizonweb', 'registryId': '789187835508', 'repositoryName': 'horizonweb', 'repositoryUri': '789187835508.dkr.ecr.eu-west-2.amazonaws.com/horizonweb', 'createdAt': '2023-06-08T14:31:25+01:00', 'imageTagMutability': 'IMMUTABLE', 'imageScanningConfiguration': {'scanOnPush': False}, 'encryptionConfiguration': {'encryptionType': 'AES256'}}, 'failed': False}"
}

PLAY RECAP ****
**
localhost : ok=20    changed=10    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```

FIGURE 3.10 – Résultat de playbook.

3.2.2 Configuration de cluster local

Pour ce partie nous allons configuré tout les ressources local que nous besoin pour la projet.

3.2.2.1 Provisionnement de les machines virtuelles

Nous allons configurer les machines virtuelles pour enfin installer le cluster local et déployé les outils nécessaire pour la fonctionnement de projet.

- La création des virtuelles machines sera faite avec Vmware Workstation (voir figure 5.11)

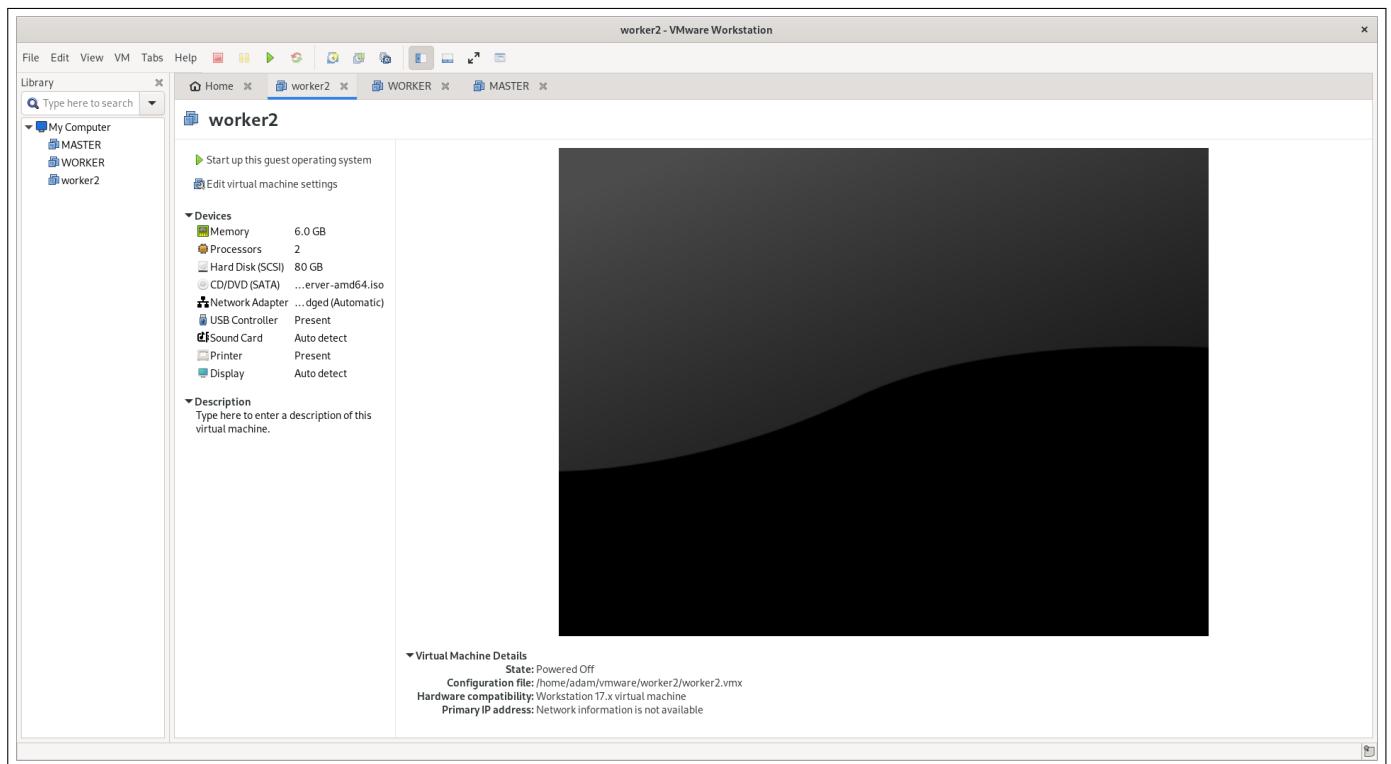


FIGURE 3.11 – Les machines virtuelles.

Pour la configuration de cluster nous avons utilisé Ansible pour l’optimisation du temps :

- L’installation des logiciels nécessaire sera fait avec ansible on exécutent le playbook "Configuration.yaml" sur les machines virtuelles.(voir figure 5.12, 5.13 et 5.14)

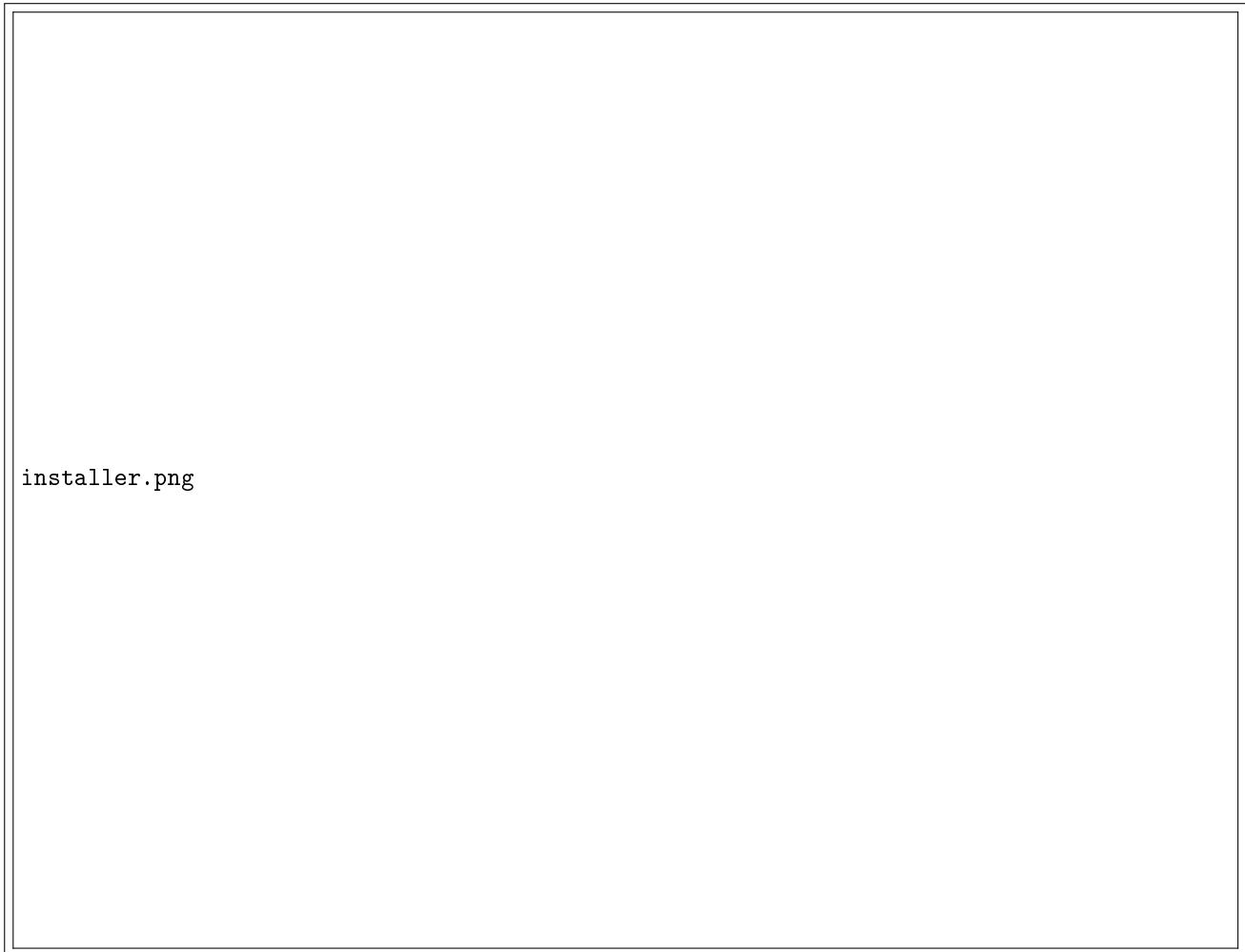


FIGURE 3.12 – Playbook Ansible d’installation(1).



FIGURE 3.13 – Playbook Ansible d’installation(2).



FIGURE 3.14 – Playbook Ansible d’installation(3).

Le résultat de l’exécution de ce playbook après la commande
"ansible-playbook Configuration.yaml"(voir figure 5.15)



resultatwhit.png

FIGURE 3.15 – résultat d'installation.

- Après l'installation, la configuration de la machine master sera fait avec playbook "ConfigMaster.yaml"(voir figure 5.156



FIGURE 3.16 – playbook Configuration de master.

Résultats de l'exécution de la playbook "ConfigMaster.yaml"
(voir figure 5.17)



CONF.png

FIGURE 3.17 – Résultat de playbook "ConfigurationMaster.yaml".

- Le cluster requis une configuration de réseau pour la communication entre les noeuds et les pods pour configurer le réseau nous avons préparer un playbook "NetworkPolicy.yaml" (voir figure 5.18)

```
1 - hosts: localhost
2   become: yes
3   become_user: root
4   tasks:
5     - name: Create Pod Network
6       become: yes
7       shell: "{{ item }}"
8       with_items:
9         - kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/tigera-operator.yaml
10        - kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/custom-resources.yaml
11     - pause: seconds=30
12     - name: Get the token for joining the nodes with Kuberentes master.
13       shell: kubeadm token create --print-join-command
14       register: kubernetes_join_command
15     - debug:
16       msg: "{{ kubernetes_join_command.stdout }}"
17
```

FIGURE 3.18 – playbook NetworkPolicy.yaml.

Résultats de l'exécution de la playbook "NetworkPolicy.yaml"
(voir figure 5.19)

```

PLAY [localhost] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [Create Pod Network] ****
changed: [localhost] => (item=kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/tigera-operator.yaml)
changed: [localhost] => (item=kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/custom-resources.yaml)

TASK [pause] ****
Pausing for 30 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [localhost]

TASK [Get the token for joining the nodes with Kuberentes master.] ****
changed: [localhost]

TASK [debug] ****
ok: [localhost] => {
    "msg": "kubeadm join 192.168.1.5:6443 --token jtv153.p99lsz6sd8nws05l --discovery-token-ca-cert-hash sha256:10499d2ea22552c16ad59a7efb287aaaf76e2931fa512c7e057c554a63a8e14bc "
}

PLAY RECAP ****
localhost : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

```

FIGURE 3.19 – Résultat de playbook "NetworkPolicy.yaml".

- Après la configuration de calico dans le cluster l'ingénieur DevOps exécute la commande "Kubeadm join" qui est afficher après l'exécution de playbook "NetworkPolicy.yaml", dans les noeud qui nous avons créé comme des noeuds du travail.(voir figure 5.19 pour le commande "Kubeadm join").
Enfin la cluster local est prêt(voir figure 5.20)

```
[adam@fedora ~]$ kubectl get pods -n kube-system
NAME                               READY   STATUS    RESTARTS   AGE
coredns-5d78c9869d-4tx79          1/1     Running   1 (30h ago)  12d
coredns-5d78c9869d-j4xkg          1/1     Running   1 (30h ago)  12d
etcd-master                         1/1     Running   1 (30h ago)  12d
kube-apiserver-master             1/1     Running   1 (30h ago)  12d
kube-controller-manager-master    1/1     Running   2 (30h ago)  12d
kube-proxy-4jchw                  1/1     Running   1 (30h ago)  12d
kube-proxy-c4cx8                  1/1     Running   1 (30h ago)  12d
kube-proxy-t97hs                  1/1     Running   2 (30h ago)  12d
kube-scheduler-master             1/1     Running   2 (30h ago)  12d
kube-state-metrics-79cd9845d7-lqdgb 1/1     Running   1 (30h ago)  12d
[adam@fedora ~]$
```

FIGURE 3.20 – Cluster locale.

3.2.2.2 Configuration de Sonarqube

Nous allons configurer et déployé SonarQube sur notre cluster pour analyse la qualité de notre code .

- Pour déployé SonarQube il faut créer les différent fichier nécessaire (voir figure 5.21)

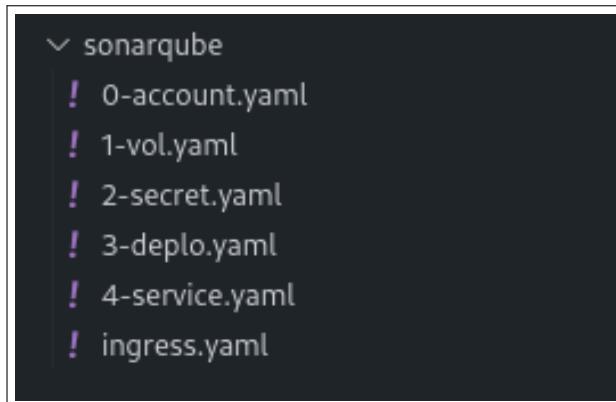


FIGURE 3.21 – Liste des fichier (SonarQube).

Ces fichier sont déployé après l'exécution de commande : "kubectl apply -f ./sonarqube -n sonarqube" (voir figure 5.22)

```
master@master:~$ kubectl get pods -n sonarqube
NAME                  READY   STATUS    RESTARTS   AGE
sonar-db9668ff8-c2z7n   1/1     Running   1 (128m ago)   10d
master@master:~$ █
```

FIGURE 3.22 – Pod SonarQube.

La figure suivante présente l’interface de SonarQube (voir figure 5.23)

Project	Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
Moovi-master	6 (C)	1 (E)	0.0% (E)	53 (A)	0.0%	3.5%	1.3k (JavaScript...)
NodeAction	4 (B)	0 (A)	0.0% (E)	8 (A)	0.0%	2.4%	2.5k (JavaScript...)

FIGURE 3.23 – Interface de SonarQube.

3.2.2.3 Configuration de Nexus

Nous allons configurer et déployer Nexus sur notre cluster stocker les artefacts de notre code.

- Pour déployer Nexus il faut créer les différents fichiers nécessaires (voir figure 5.24)

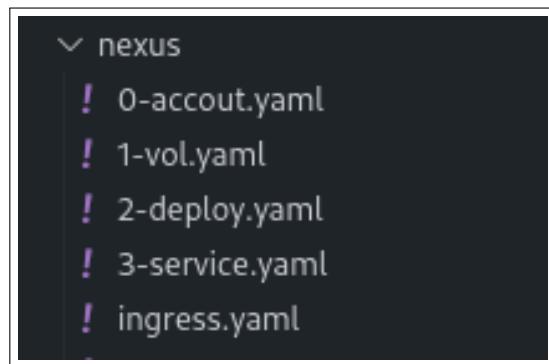


FIGURE 3.24 – Liste des fichiers (Nexus).

Ces fichiers sont déployés après l'exécution de la commande : "kubectl apply -f ./nexus -n nexus" (voir figure 5.25)

```
master@master:~$ kubectl get pods -n nexus
NAME           READY   STATUS    RESTARTS   AGE
nexus-698c45f867-h6grg   1/1     Running   1 (127m ago)   10d
master@master:~$ █
```

FIGURE 3.25 – Pod Nexus.

La figure suivante présente l'interface de Nexus (voir figure 5.26) :

L'interface de l'application Sonatype Nexus Repository OSS 3.55.0-01. Le menu principal à gauche indique 'Browse' en vert. La partie centrale montre une liste de fichiers compressés ('tgz') nommés 'horizon-0.1.0.tgz', 'horizon-0.1.1.tgz' et 'horizon-1.0.0.tgz' dans un dossier 'horizon'.

FIGURE 3.26 – Interface de Nexus.

3.2.2.4 Outils des surveillances

- Pour déployer les outils de surveillance il faut créer les différents fichiers nécessaires (voir figure 5.27)



FIGURE 3.27 – Liste des fichiers (Monitoring).

Ces fichiers sont déployés après l'exécution de la commande : "kubectl apply -f ./Monitoring -n monitoring" (voir figure 5.28)

```

master@master:~$ kubectl get pods -n monitoring
NAME                      READY   STATUS    RESTARTS   AGE
alertmanager-6ccfbf545c-glgrf   1/1     Running   1 (127m ago)   9d
grafana-84b5bd4fb4-9q2tb       1/1     Running   1 (127m ago)   10d
prometheus-deployment-875d8fb4b-zsw95   1/1     Running   3 (29m ago)   30m
master@master:~$ █
  
```

FIGURE 3.28 – Les Pods du namespace Monitoring.

error : unable to upgrade connection : container not found ("grafana")

3.3 Sprint2 :Configuration de pipeline

Pour ce partie nous allons configurer les workflow qui sont lancer quand une changement dans notre code est détecter.

3.3.1 Crédation des pipelines

Configuration de pipeline pour les différent branches :

- Pipeline de branche Main (voir figure 5.29 et 5.30)

```

name: Build
on:
  push:
    branches:
      - main
jobs:
  build-and-deploy:
    name: Build
    runs-on: "self-hosted"
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0 # Shallow clones should be disabled for a better relevancy of analysis
      - uses: sonarsource/sonarqube-scan-action@master
        env:
          SONAR_TOKEN: ${{ secrets.SONARQUBE_TOKEN }}
          SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}
      - uses: sonarsource/sonarqube-quality-gate-action@master
        timeout-minutes: 5
        env:
          SONAR_TOKEN: ${{ secrets.SONARQUBE_TOKEN }}

      - name: Disable treating warnings as errors
        run: echo "CI=false" >> $GITHUB_ENV
      - name: Get application version
        id: get-version
        run: echo ::set-output name=version::$(node -p "require('./package.json').version")"

      - name: Use Node.js
        uses: actions/setup-node@v3
      - run: npm ci --legacy-peer-deps
      - run: npm run build --if-present
      - name: Build artifact
        run: npm pack
      - name: Publish NPM package to Nexus
        run: npm publish ./horizon-${{ steps.get-version.outputs.version }}.tgz --registry=${{ secrets.NEXUS_REPO }}
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v2 # More information on this action can be found below in the 'AWS Credentials' section
        with:
          aws-region: eu-west-2
      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1

      - name: Build and push Docker image
        run:
          docker build -t ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.eu-west-2.amazonaws.com/horizonweb:${{ steps.get-version.outputs.version }} .
          docker push ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.eu-west-2.amazonaws.com/horizonweb:${{ steps.get-version.outputs.version }} .

```

FIGURE 3.29 – Pipeline de branche main(1).

```

- name: Create Namespace if not exists
  id: create-namespace
  run:
    namespace="webapp"
    if ! kubectl get namespace "${namespace}" --output=name 2>/dev/null; then
      echo "Namespace does not exist. Creating..."
      kubectl create namespace "${namespace}"
      echo "::set-output name=namespace::${namespace}"
    else
      echo "Namespace already exists."
      echo "::set-output name=namespace::${namespace}"
    fi

- name: Deploy to Local Cluster
  env:
    NAMESPACE: ${{ steps.create-namespace.outputs.namespace }}
    IMAGE_REFERENCE: ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.eu-west-2.amazonaws.com/horizonweb:${{ steps.get-version.outputs.version }}
  run:
    aws ecr get-login-password --region eu-west-2 | docker login --username AWS --password-stdin ${{ secrets.AWS_ACCOUNT_ID }}.dkr.ecr.eu-west-2.amazonaws.com
    kubectl config use-context ${{ secrets.LOCAL_CONTEXT }}
    secret="my-registry-key"
    if ! kubectl get secret "${secret}" -n ${NAMESPACE} 2>/dev/null; then
      echo "Secret does not exist. Creating..."
      kubectl create secret generic "${secret}" --from-file=.dockerconfigjson=/home/runner/.docker/config.json --type=kubernetes.io/dockerconfigjson -n ${NAMESPACE}
    elif kubectl get secret "${secret}" -n ${NAMESPACE} 2>/dev/null; then
      echo "Secret already exists."
      kubectl delete secret "${secret}" -n ${NAMESPACE}
      kubectl create secret generic "${secret}" --from-file=.dockerconfigjson=/home/runner/.docker/config.json --type=kubernetes.io/dockerconfigjson -n ${NAMESPACE}
    fi

    kubectl apply -f ./kubernetes/Local_test/ServiceAccount.yaml -n ${NAMESPACE}
    kubectl apply -f ./kubernetes/Local_test/Volume.yaml -n ${NAMESPACE}
    cat ./kubernetes/Local_test/Delyoement.yaml | envsubst | kubectl apply -n ${NAMESPACE} -f -
    kubectl apply -f ./kubernetes/Local_test/service.yaml -n ${NAMESPACE}

```

FIGURE 3.30 – Pipeline de branche main(2).

Encore de la lancement de la pipeline différent action sera exécuter (voir figure 5.31)

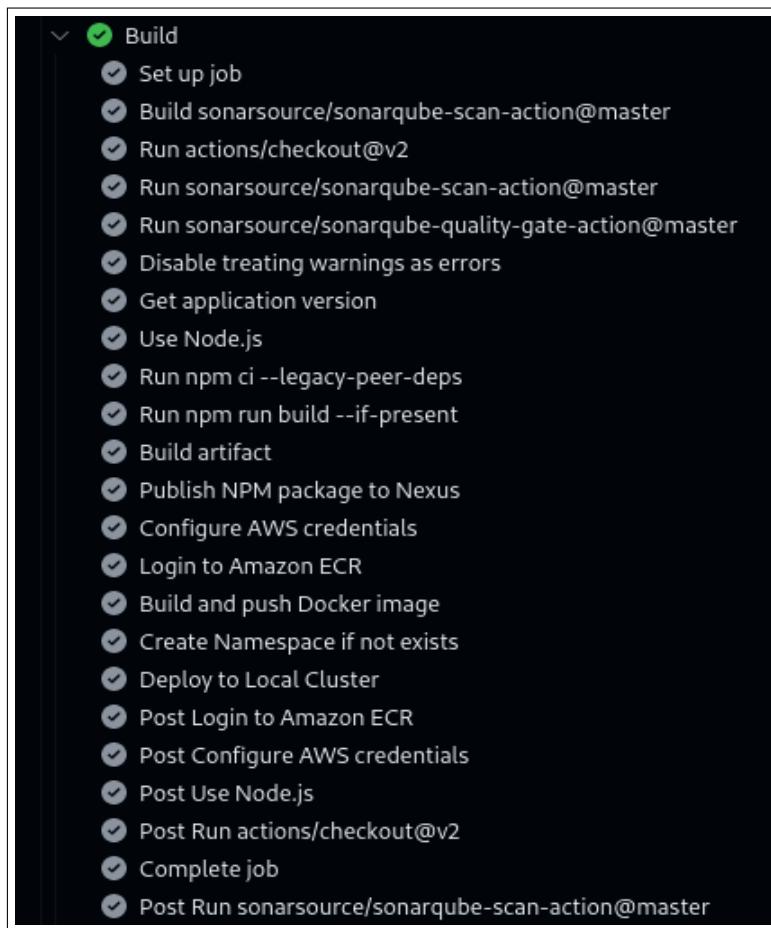


FIGURE 3.31 – Github Action interface.

- Pipeline de branche EKS (voir figure 5.32 et 5.33)

```

name: Build
on:
  push:
    branches:
      - EKS
jobs:
  build-and-deploy:
    name: Build
    runs-on: "self-hosted"
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0 # Shallow clones should be disabled for a better relevancy of analysis
      - uses: sonarsource/sonarqube-scan-action@master
        env:
          SONAR_TOKEN: ${{ secrets.SONARQUBE_TOKEN }}
          SONAR_HOST_URL: ${{ secrets.SONAR_HOST_URL }}
      - uses: sonarsource/sonarqube-quality-gate-action@master
        timeout-minutes: 5
        env:
          SONAR_TOKEN: ${{ secrets.SONARQUBE_TOKEN }}
      - name: Disable treating warnings as errors
        run: echo "CI=false" >> $GITHUB_ENV
      - name: Get application version
        id: get-version
        run: echo "::set-output name=version::$(node -p "require('./package.json').version")"
      - name: Use Node.js
        uses: actions/setup-node@v3
      - run: npm ci --legacy-peer-deps
      - run: npm run build --if-present
      - name: Build artifact
        run: npm pack
      - name: Publish NPM package to Nexus
        run: npm publish ./horizon-${{ steps.get-version.outputs.version }}.tgz --registry=${{ secrets.NEXUS_REPO }}
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v2 # More information on this action can be found below in the 'AWS Credentials' section
        with:
          aws-region: eu-west-2
      - name: Login to Amazon ECR
        id: login-ecr
        uses: aws-actions/amazon-ecr-login@v1
      - name: Build and push Docker image
        run:
          docker build -t ${secrets.AWS_ACCOUNT_ID}.dkr.ecr.eu-west-2.amazonaws.com/horizonweb:latest .
          docker push ${secrets.AWS_ACCOUNT_ID}.dkr.ecr.eu-west-2.amazonaws.com/horizonweb:latest
          echo "Get kubeconfig file"

```

FIGURE 3.32 – Pipeline de branche EKS(1).

```

- name: Get kubeconfig file
  run: aws eks update-kubeconfig --region eu-west-2 --name EKS-CLUSTER
- name: Create Namespace if not exists
  id: create-namespace
  run:
    - namespace="webapp"
      if ! kubectl get namespace "${namespace}" --output=name 2>/dev/null; then
        echo "Namespace does not exist. Creating..."
        kubectl create namespace "${namespace}"
        echo "::set-output name=namespace::${namespace}"
      else
        echo "Namespace already exists."
        echo "::set-output name=namespace::${namespace}"
      fi
- name: Deploy to EKS Cluster
  env:
    NAMESPACE: ${steps.create-namespace.outputs.namespace}
    IMAGE_REFERENCE: ${secrets.AWS_ACCOUNT_ID}.dkr.ecr.eu-west-2.amazonaws.com/horizonweb:latest
  run:
    - kubectl config use-context ${secrets.EKS_CONTEXT}
    - kubectl apply -f ./kubernetes/EKS_Deploement/ServiceAccount.yaml -n $NAMESPACE
    - kubectl apply -f ./kubernetes/EKS_Deploement/Volume.yaml -n $NAMESPACE
    - cat ./kubernetes/EKS_Deploement/Delyoement.yaml | envsubst | kubectl apply -n $NAMESPACE -f -
    - kubectl apply -f ./kubernetes/EKS_Deploement/service.yaml -n $NAMESPACE

```

FIGURE 3.33 – Pipeline de branche EKS(2).

Encore de la lancement de la pipeline différent action sera exécuter (voir figure 5.34)

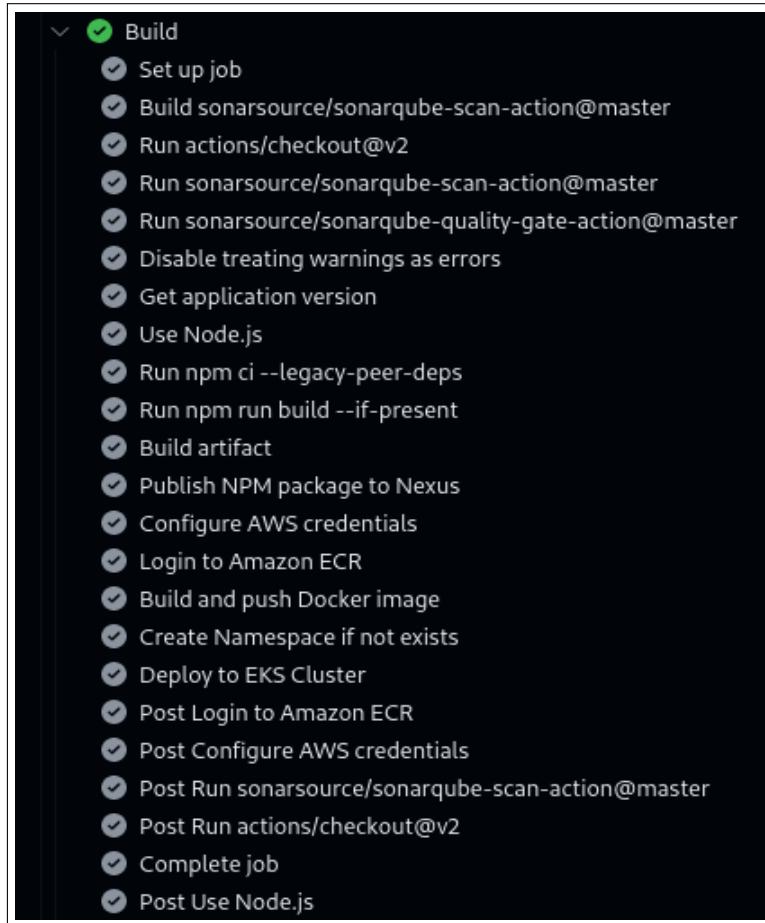


FIGURE 3.34 – Github Action interface.

3.3.2 Configuration d'exécuteur local

Pour ce partie nous allons configuré l'exécuteur qui exécuter les workflows de notre projet.(voir figure 5.35)

```
  /----( )----|----|----|----|----|----|----|----|----|----|----|----|----|----|
 |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
 |----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
 \----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
   /----\----|----|----|----|----|----|----|----|----|----|----|----|----|----|
  /----\----|----|----|----|----|----|----|----|----|----|----|----|----|----|
  /----\----|----|----|----|----|----|----|----|----|----|----|----|----|----|
 /----\----|----|----|----|----|----|----|----|----|----|----|----|----|----|
/----\----|----|----|----|----|----|----|----|----|----|----|----|----|----|
\----\----|----|----|----|----|----|----|----|----|----|----|----|----|----|
\----\----|----|----|----|----|----|----|----|----|----|----|----|----|----|
Self-hosted runner registration
```

```
# Authentication

Using V2 flow: False

✓ Connected to GitHub

# Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]

Enter the name of runner: [press Enter for runner]

This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

✓ Runner successfully added
✓ Runner connection is good

# Runner settings

Enter name of work folder: [press Enter for _work]

✓ Settings Saved.
```

FIGURE 3.35 – Github Action Runner.

Aprés l'exécution d'un pipeline l'application sera deployé dans l'environnement corespond a la besoin.

- Le figure suivante présente l'application exécuter par le pipline dans l'environnement test (voir figure 5.36) :

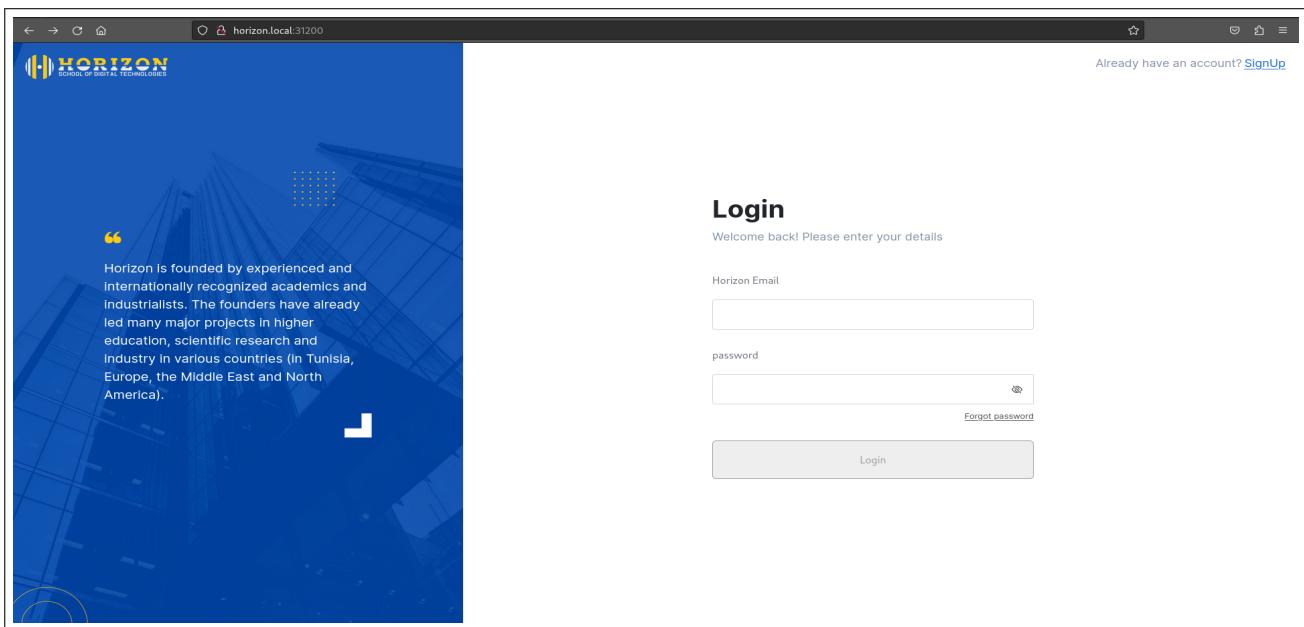


FIGURE 3.36 – Application dans environnement test.

L'image docker de cette version d'application sera stocker dans ECR (voir figure 5.37)

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Scan status	Vulnerabilities
0.1.1	Image	June 09, 2023, 09:13:54 (UTC+01)	56.41	Copy URI	sha256:de6072878cb2fc54f7e22ca7d80791...	-	-

FIGURE 3.37 – Répertoire des images ECR.

- La figure suivante présente l'application exécuter par le pipeline dans l'environnement prod :

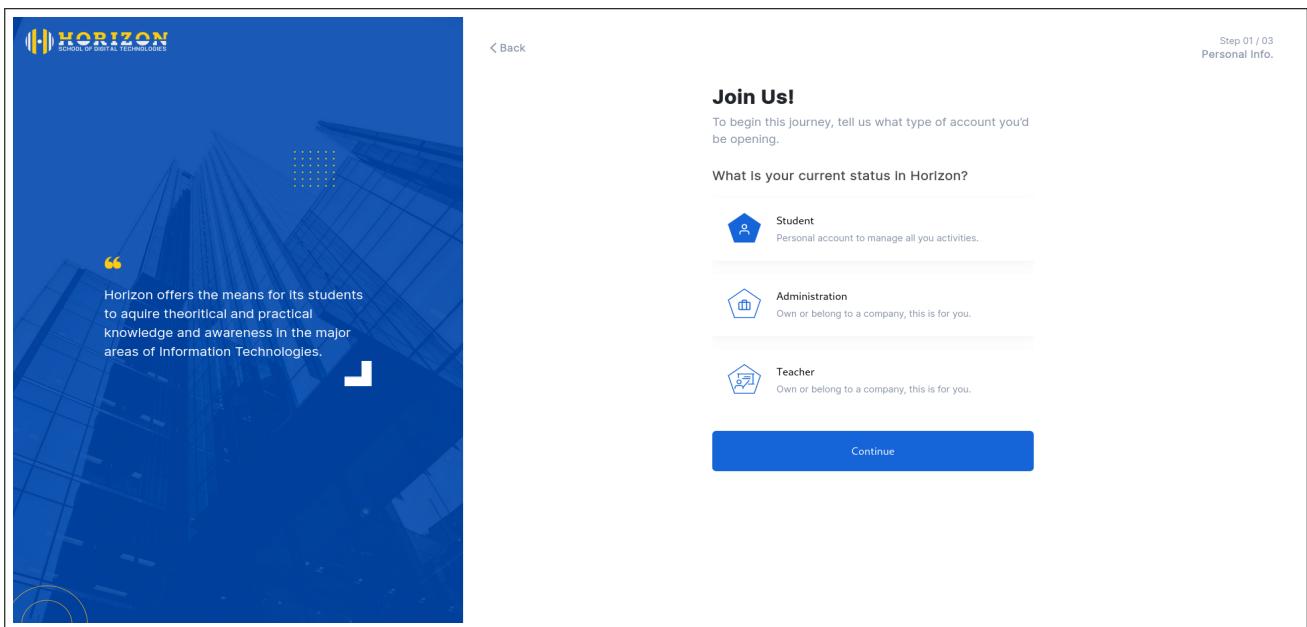


FIGURE 3.38 – Application dans environnement production.

L'image docker de cette version d'application sera stocker dans ECR (voir figure)

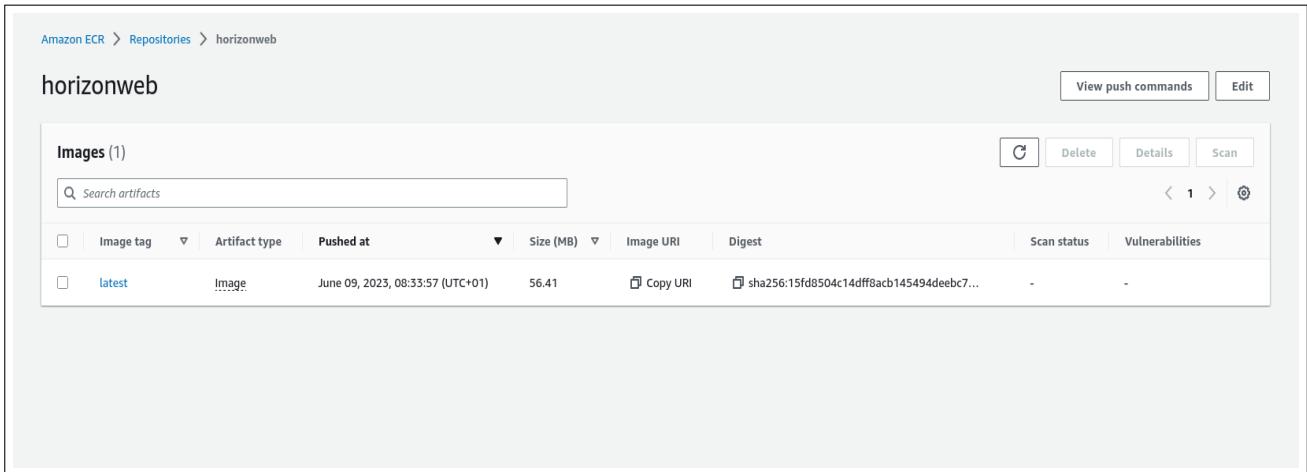


FIGURE 3.39 – Image latest dans ECR.

3.4 Sprint3 : Mise en place d'un système du monitoring

Nous avons configuré et déployé des surveillances pour collecter, analyser et stocker les données de notre cluster.

3.4.1 Configuration de Prometheus

- Interface de Prometheus(voir figure 5.40)

The screenshot shows the Prometheus 'Targets' page. At the top, there are navigation links: 'Prometheus', 'Alerts', 'Graph', 'Status', 'Help', and a user icon. Below the navigation is a search bar with a placeholder 'Filter by endpoint or labels'. To the right of the search bar are three checkboxes: 'Unknown' (checked), 'Unhealthy' (checked), and 'Healthy' (unchecked). The main content area is titled 'Targets' and lists several service instances with their status and uptime. Each entry includes a 'show more' link.

Service	Status
kube-state-metrics	1/1 up
kubernetes-apiservers	1/1 up
kubernetes-cadvisor	3/3 up
kubernetes-nodes	3/3 up
kubernetes-service-endpoints	6/6 up
nexus	1/1 up
sonarqube	1/1 up

FIGURE 3.40 – Interface Prometheus.

3.4.2 Configuration de AlertManager

- Interface de AlertManager(voir figure 5.41)

The screenshot shows the AlertManager 'Alerts' page. At the top, there are navigation links: 'Alertmanager', 'Alerts', 'Silences', 'Status', 'Settings', 'Help', and a 'New Silence' button. Below the navigation is a search bar with 'Filter' and 'Group' tabs. To the right of the search bar are buttons for 'Receiver: All', 'Silenced', and 'Inhibited'. The main content area lists alerts. One alert is shown in detail: 'alertname="POD EVICTED"'. Below the alert name are buttons for 'Info', 'Source', 'Silence', and 'Link'. At the bottom of the alert card are several filter tags: '_alert_rule_namespace_uid_="f096f41e-0125-42d3-81c3-dac10efef6f"', '_alert_rule_uid_="aeb3e2cf-9869-4131-8e7b-095e1fcace74"', 'datasource_uid="ed657bb4-697a-4634-8432-cb388058697f"', 'grafana_folder="CLUSTER_LOCAL"', 'ref_id="A"', and 'severity="critical"'. There is also a '+ Silence' button.

FIGURE 3.41 – Interface AlertManager.

- Nous avons configuré aussi Slack pour recevoir les notifications des alertes (voir figure 5.42)

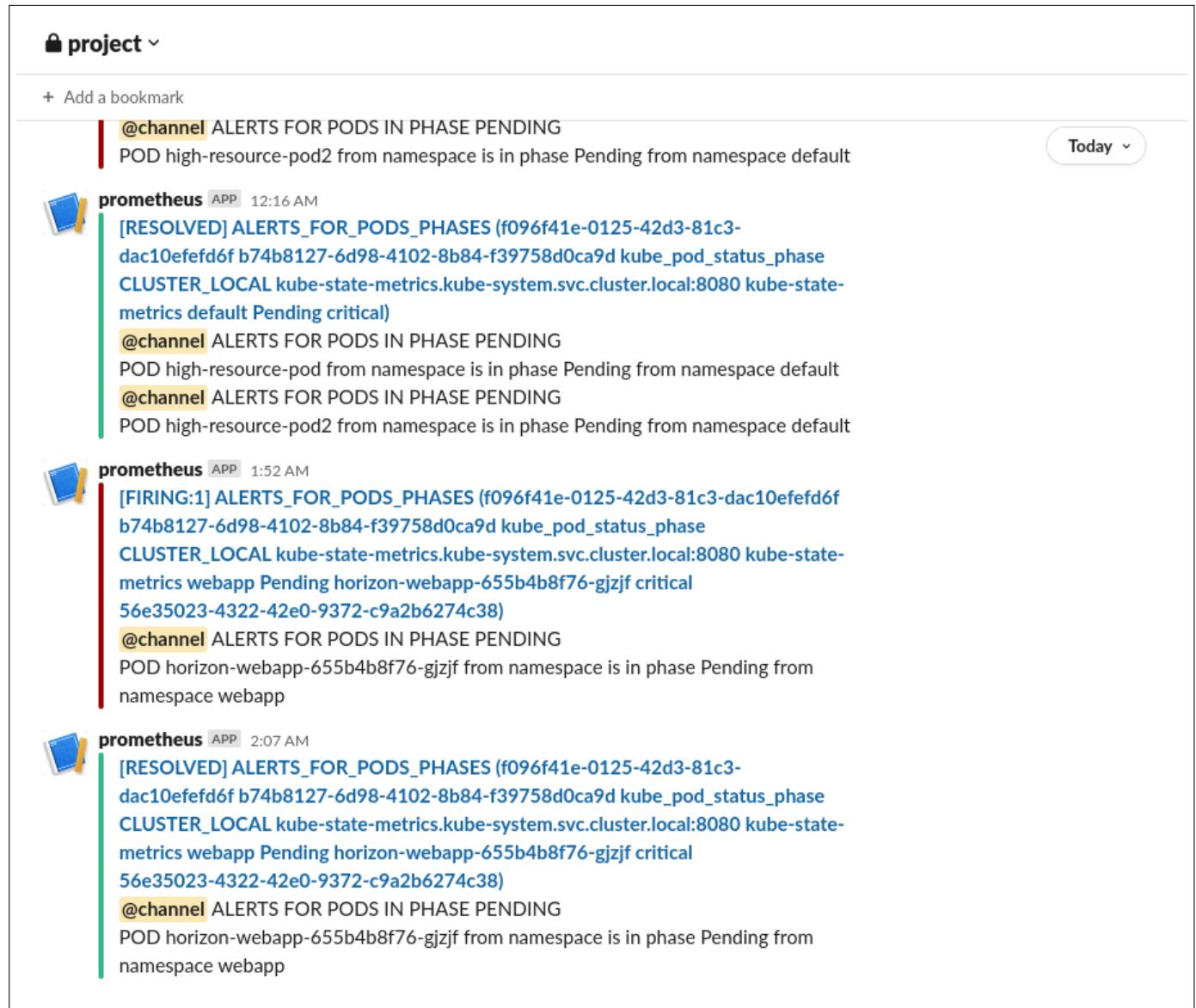


FIGURE 3.42 – Interface de Slack.

3.4.3 Configuration des Dashboards Grafana

Nous avons configuré les différent dashboards qui nous allons utiliser pour la visualisation des données et métriques collecter par prometheus.

- Interface de Grafana(voir figure 5.43)

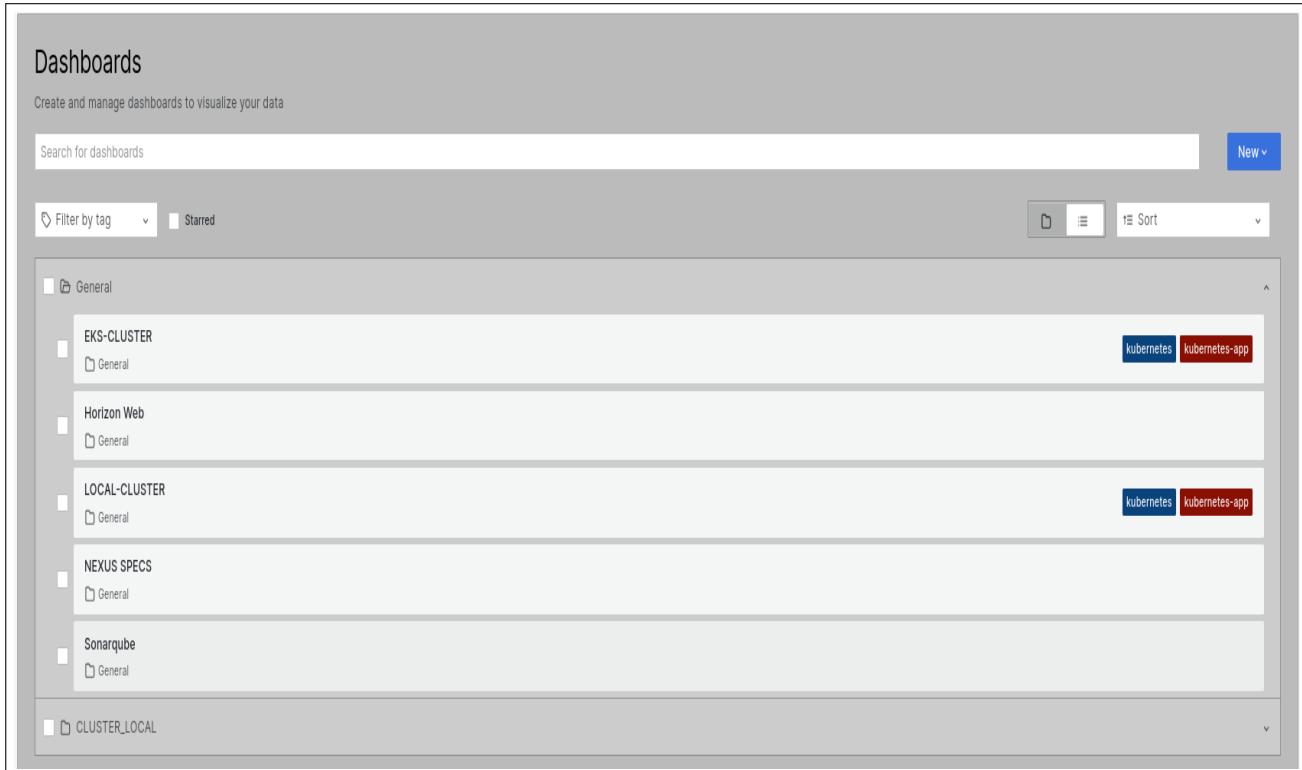


FIGURE 3.43 – Interface Grafana.

Dans les figures suivante nous allons présenté différent exemple des Dashboard :

- Exemple : dashboard de Local cluster(voir figure 5.44)

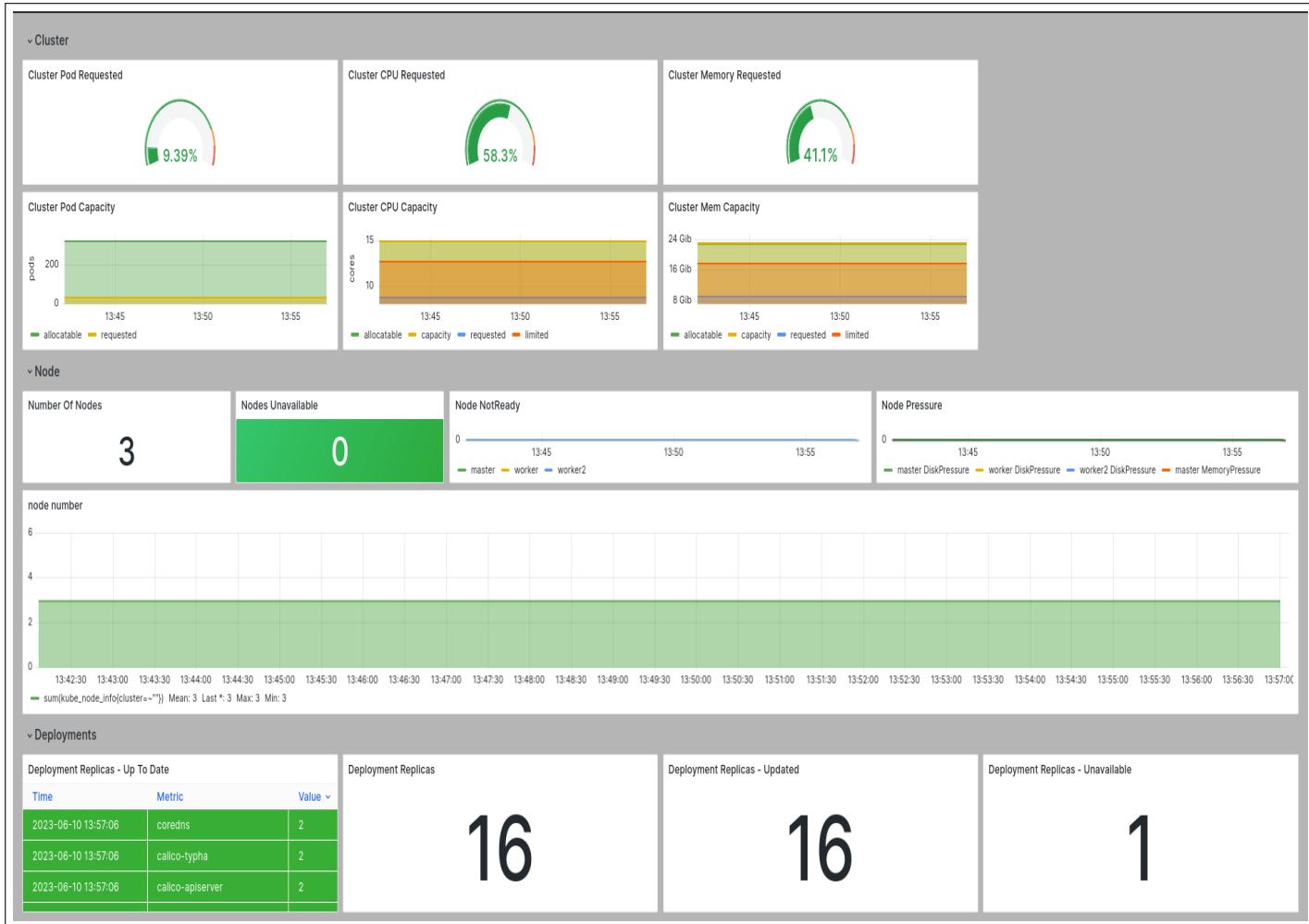


FIGURE 3.44 – dashboard de Local cluster.

- Exemple : dashboard de SonarQube(voir figure 5.45)

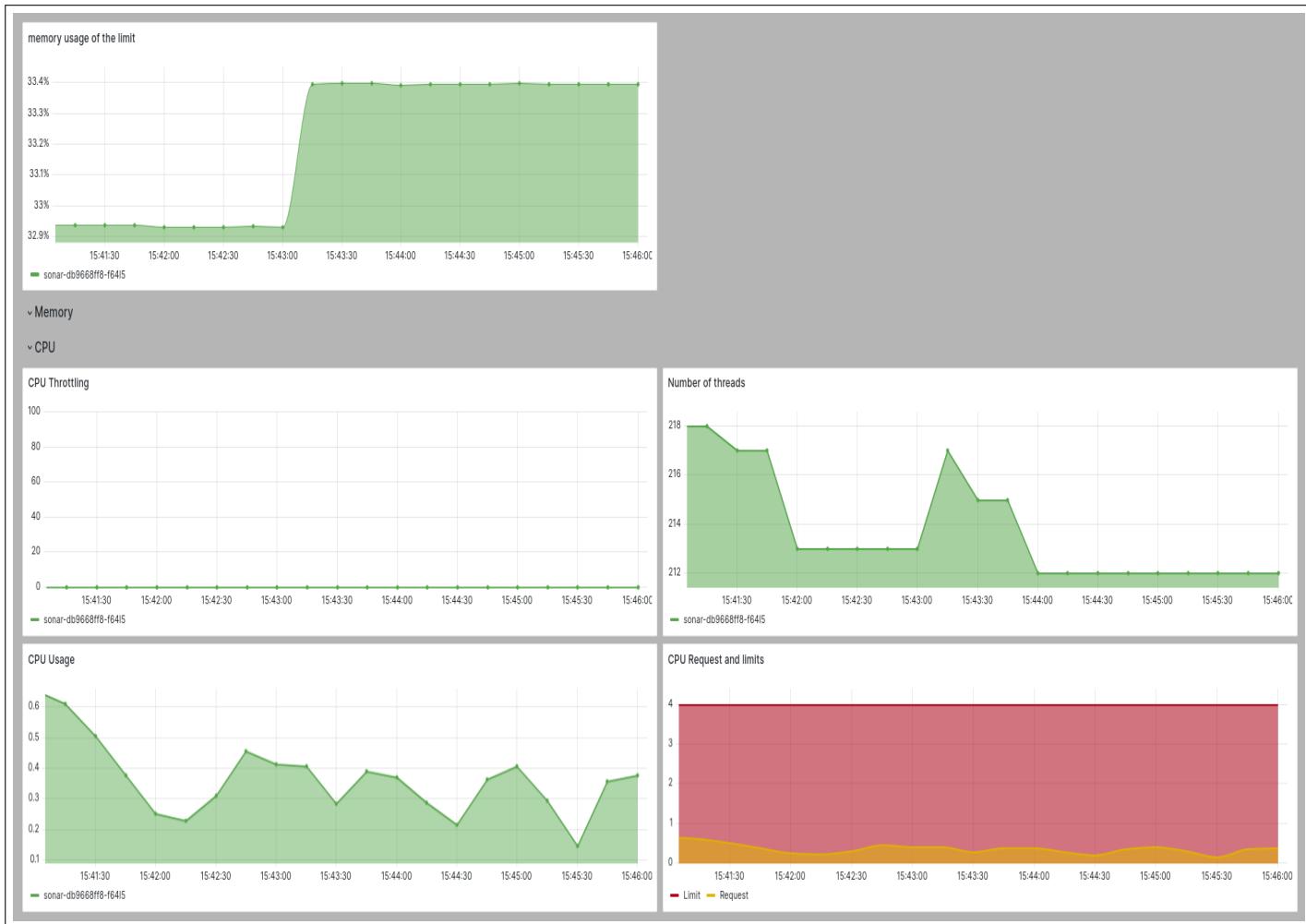


FIGURE 3.45 – dashboard de SonarQube.

- Exemple : dashboard de EKS cluster(voir figure 5.46)

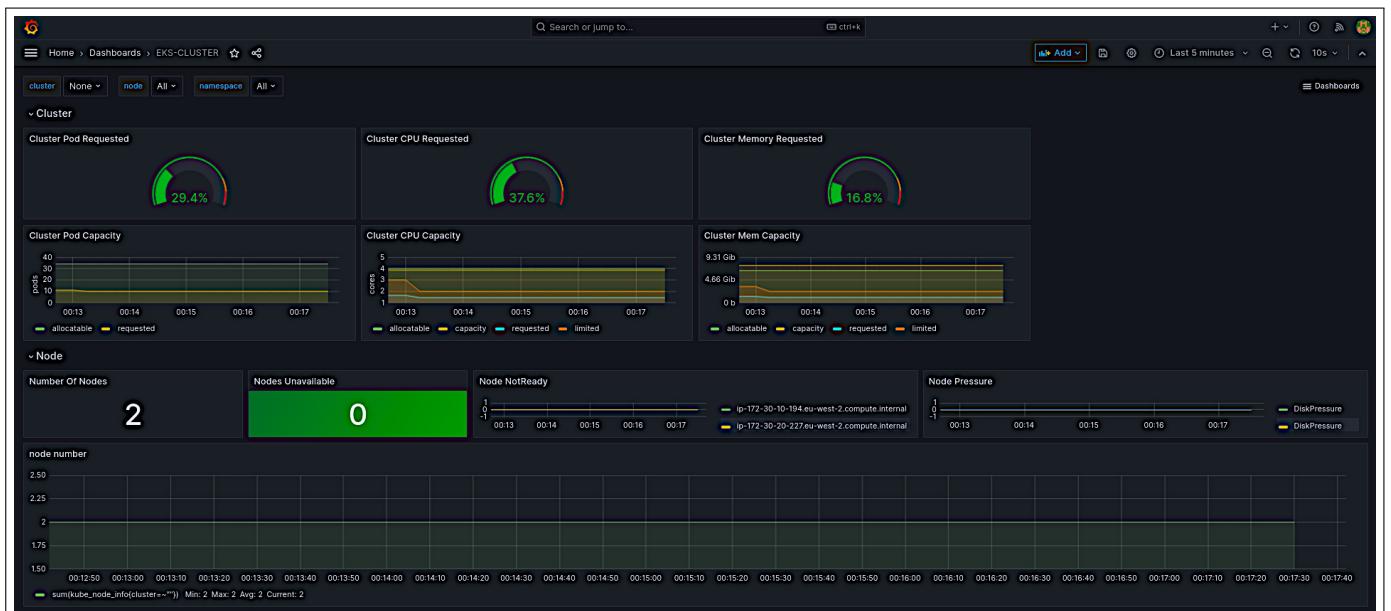


FIGURE 3.46 – dashboard de EKS cluster.

3.5 Conclusion

Dans ce chapitre nous avons entamé la réalisation du projet tout en insistant sur quelques scénarios qui montrent les fonctionnalités majeures de l'application.

Bibliographie

- [1] Microsoft Azure. Définition de devops. [Online ; accessed 01-mars-2023].
- [2] Microsoft Azure. Définition de cloud. [Online ; accessed 01-mars-2023].
- [3] Oracle. Définition saas. [Online ; accessed 01-mars-2023].
- [4] Oracle. Définition de paas. [Online ; accessed 01-mars-2023].
- [5] Microsoft Azure. Définition de iaas. [Online ; accessed 01-mars-2023].
- [6] Oracle. Définition de cloud prive. [Online ; accessed 01-mars-2023].
- [7] Google. Définition de public cloud. [Online ; accessed 01-mars-2023].
- [8] Goggle. Définition de cloud hybride. [Online ; accessed 01-mars-2023].
- [9] Microsoft Azure. Azure devops. [Online ; accessed 01-mars-2023].
- [10] Amazon web service. Amazon devops. [Online ; accessed 01-mars-2023].
- [11] Bluelight. Gcp devops. [Online ; accessed 01-mars-2023].
- [12] Wikipedia. Définition d'acteur dans un système. [Online ; accessed 01-mars-2023].
- [13] Wikipedia. Diagramme de séquence. [Online ; accessed 01-mars-2023].
- [14] Wikipedia. Diagramme de classe. [Online ; accessed 01-mars-2023].
- [15] Wikepedia. Diagramme de déploiement. [Online ; accessed 01-mars-2023].