

2024/2025



HONORIS UNITED UNIVERSITIES

RAPPORT DE STAGE D'IMMERSION EN ENTREPRISE

SUJET: Conception et Développement des Microservices avec Spring Cloud en Implémentant une Stratégie CI/CD et une Orchestration pour Gérer la Surcharge de Trafic

Réalisé par: Ben Ammar Fares

Encadré par: Mr. NAOUAR Kadhem



Effectué au sein de: AMI Assurance
Du 04-07-2024 Au 04-09-2024

Remerciements

Je voudrais profiter de ces quelques lignes et de cette occasion qui m'est donnée de le faire pour remercier toutes les personnes qui, de près ou de loin, ont permis la réalisation de ce stage dans les meilleures conditions.

J'adresse ainsi mes plus sincères remerciements à Monsieur Kadhem Naouar , pour sa confiance, son écoute et son accompagnement dont j'ai pu bénéficier tout au long de ce stage. Grâce à lui, j'ai pu affiner mon projet professionnel et acquérir de solides et nouvelles compétences pour mon futur.

J'ai également une pensée chaleureuse pour mes proches, membres de ma famille et amis, qui n'ont cessé de me soutenir et de m'encourager au quotidien dans toutes mes démarches, à la fois sur le plan professionnel et sur le plan plus personnel.

Cette expérience m'a motivé à être plus innovant et à travailler dur pour atteindre mes objectifs .

Table de matière

Introduction Générale	1
CHAPITRE 1: Présentation de l'entreprise.....	3
1.1 Contexte du projet	4
1.2 Problématique	5
1.3 Solution proposée	5
CHAPITRE 2: Qu'est-ce que le Cloud Computing.....	6
2.1 Les Environnements de Cloud	7
• Modèles de déploiement du cloud	7
• La différence entre les modèles	8
• Principaux services cloud	10
2.2 Avantages du Cloud Computing.....	11
2.3 Inconvénient du Cloud Computing	12
CHAPITRE 3: Qu'est ce que c'est le concept de virtualisation	13
3.1 Avant la virtualisation	13
3.2 Définition	14
3.3 Petite histoire sur la virtualisation	16
3.4 Fonctionnement	17
• Hyperviseur	17
• Difference entre les deux types	20
• Fonctionnement de la virtualisation	21
3.5 Types de ressources virtualisées	22
3.6 Machines virtuelles	23
• Définition	23
• Avantages.....	24
• Lien entre les machines virtuelles et le cloud	24
3.7 Conteneurs	25
• Qu'est-ce que Docker	25
• Les services Containers as a Service (CaaS)	26
• Docker et les services de conteneurs	27

3.8 Kubernetes	27
• Fonctionnalités et Utilité	27
• L'architecture du Kubernetes	28
• Les principaux concepts liés à Kubernetes	32
• Types de services Kubernetes	34
• Les Objets Kubernetes	35
3.8 Minikube	35
• Qu'est-ce que Minikube	35
• Caractéristiques Clés du Minikube concernant le Testing	36
• Limitations de Minikube concernant le déploiement	37
CHAPITRE 4: Automatisation	38
4.1 Jenkins	38
• Qu'est-ce que Jenkins	38
• Caractéristiques clés de Jenkins	38
• Limitations de Jenkins	39
CHAPITRE 5: Analyse des services	40
5.1 Sonarqube	40
• Qu'est-ce que Sonarqube	40
• Fonctionnalités de SonarQube	40
• Fonctionnement	41
• Limitations de SonarQube	41
CHAPITRE 6: L'architecture Microservices	42
• Concept des microservices	42
• Caractéristiques clés des microservices	42
• Architecture et Fonctionnement	42
• Bonnes Pratiques et Outils	43
• Principe de Découplage	43
• Limitations des microservices	44
• Différence entre l'architecture microservice et monolithique	45

CHAPITRE 7: Conception et Réalisation	46
• Partie Développement avec Spring Cloud et Microservices	46
• Partie Dockerisation des microservices	47
• Partie Communication entre les conteneurs	48
• Partie Orchestration des conteneurs	50
• Partie préparation du serveur SonarQube	57
• Partie Intégration avec Jenkins	58
• Partie Intégration avec Minikube	60
• Partie Automatisation	64
• Partie Mise à l'échelle des services	71
• Conclusion	72



Introduction Générale

Lorsque nous développons une application, nous suivons toujours le même processus pour la livrer à l'utilisateur final, et c'est l'objectif principal, que nous utilisions une méthodologie Agile comme Scrum, XP ou une méthodologie Lourde comme 2TUP, RUP, etc.

Comme nous le savons, le processus typique de publication de logiciels commence par une idée qui génère des exigences, qui sont ensuite codées, testées et déployées sur un serveur public pour que les utilisateurs puissent y accéder. Pour cela, vous devez construire et empaqueter votre application sous une forme exécutable afin qu'elle puisse fonctionner en configurant le serveur public avec tous les outils nécessaires, en installant les dépendances et en configurant le pare-feu pour que l'application puisse être lancée et utilisée par les utilisateurs.

Cependant, le processus ne s'arrête pas là. Pendant l'utilisation, vous devrez vérifier que tout fonctionne correctement, identifier et corriger les bugs éventuels, gérer les pics de charge utilisateur, ajouter de nouvelles fonctionnalités et optimiser les performances. Le déploiement n'est donc pas la dernière étape à exécuter.

Après le lancement initial, de multiples mises à jour seront nécessaires, avec un système de versionnage (par exemple 1.1.4) à chaque nouvelle implémentation, test, construction et déploiement, suivis d'observations pour identifier les améliorations et les problèmes à résoudre rapidement. Cela crée un processus de livraison continue des changements, qui doit être efficace, rapide et avec un minimum de bugs, ce qui représente un défi de taille.

Les principaux défis sont les suivants : manque de communication et de collaboration entre les développeurs et les opérations, conflits d'intérêts entre la volonté de pousser rapidement de nouvelles fonctionnalités et le besoin de maintenir la stabilité en production, les tests de sécurité et d'application qui ralentissent le processus, et le travail manuel qui est lent et sujette aux erreurs humaines.

C'est dans ce contexte que le concept de **DevOps** a été créé, dans le but de supprimer ces obstacles et d'automatiser entièrement le processus de publication, en impliquant étroitement les développeurs et les opérations.



CHAPITRE 1: Présentation de l'entreprise

Ami Assurance est l'un des principaux fournisseurs d'assurance en Tunisie, offrant une large gamme de produits et services d'assurance aux particuliers et aux entreprises. Basée à Lac, en Tunisie, la société a été établie en 2003 et est depuis devenue un nom de confiance dans l'industrie de l'assurance tunisienne.

Ami Assurance opère à travers un réseau de bureaux régionaux et d'agences à travers le pays, offrant un accès facile à ses services pour les clients. La société propose un portefeuille diversifié de solutions d'assurance notamment :

- Assurance-vie
- Assurance générale
- Assurance maladie
- Assurance entreprise

Ami Partenaires



1.1 Contexte du projet

L'architecture des microservices représente une approche innovante dans le développement d'applications, permettant de construire des systèmes plus flexibles et résilients. Contrairement aux applications monolithiques traditionnelles, les microservices sont des unités autonomes qui se concentrent sur des fonctionnalités spécifiques, favorisant ainsi la scalabilité et la maintenance. Cette approche permet aux équipes de développer, déployer et mettre à l'échelle chaque service indépendamment, tout en choisissant les technologies les mieux adaptées à chaque cas.

Pour faciliter le développement et le déploiement de ces microservices, des frameworks tels que Spring Boot et Spring Cloud jouent un rôle clé. Spring Boot, en particulier, s'adapte parfaitement à l'architecture microservices et offre des outils puissants pour le développement dans le cloud. Spring Cloud fournit des fonctionnalités essentielles, telles que la découverte de services, la configuration centralisée et la gestion des API, qui simplifient la création d'applications distribuées.

La conteneurisation, principalement à travers Docker, est également un élément fondamental de cette architecture. Elle permet de conditionner les applications et leurs dépendances dans des conteneurs légers et portables, assurant un déploiement cohérent à travers différents environnements.

En parallèle, les pratiques d'intégration continue et de déploiement continu (CI/CD) automatisent les processus de construction, de test et de déploiement des services. Cela inclut également l'envoi des services vers des serveurs dédiés pour des analyses approfondies, garantissant ainsi la qualité et la performance des applications.

Enfin, pour que ces services soient accessibles et exploitables, ils doivent être exposés et déployés, permettant ainsi des tests et des inspections efficaces.

1.2 Problématique

La mise en œuvre d'une architecture microservices pose des défis significatifs pour les entreprises, notamment en matière de scalabilité, de résilience et de sécurité. Il est donc essentiel de développer des stratégies efficaces pour déployer et gérer ces services, tout en intégrant des outils comme Docker et en utilisant des pipelines CI/CD.

Évidemment, l'architecture microservices implique de nombreux services qui doivent être configurés, conteneurisés, analysés et déployés individuellement. Effectuer ces tâches manuellement peut créer une routine laborieuse pour les ingénieurs informatiques, rendant l'approche peu flexible et difficile à maintenir. Une gestion manuelle des services augmente également le risque d'erreurs humaines et complique l'automatisation des processus.

De plus, même après avoir atteint la phase finale de déploiement et d'exposition, tous les services peuvent rencontrer des pannes en raison d'une surcharge de trafic. Cela peut entraîner des déceptions pour les clients et nuire à la réputation de l'entreprise dans un environnement de travail compétitif. Il est donc crucial de mettre en place des mécanismes de résilience et de surveillance pour anticiper et gérer ces situations.

1.3 Solution proposé

Mon travail sera divisé en trois phases. Tout d'abord, je me concentrerai sur le développement de microservices en utilisant Spring Cloud, suivi de la conteneurisation de ces services et de leur envoi vers Docker Hub pour centraliser leur gestion.

La deuxième phase impliquera l'orchestration des conteneurs déjà développés dans la première phase, en utilisant Kubernetes, plus précisément Minikube, pour faciliter la gestion et le déploiement.

Enfin, nous mettrons en place un pipeline CI/CD afin d'automatiser les processus des premières et secondes phases. Cela garantira une flexibilité et une automatisation optimales, tout en abordant la mise à l'échelle des services comme une solution pour prévenir les pannes causées par des surcharges de trafic.

2.0 Qu'est-ce que le Cloud Computing

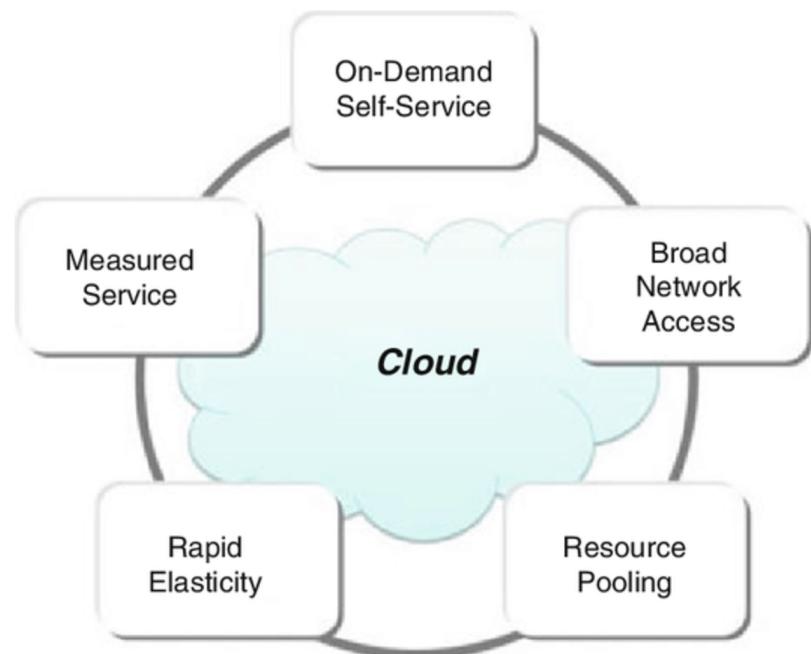
Le cloud computing est un modèle qui permet un accès réseau pratique et à la demande à un **pool partagé de ressources informatiques configurables**, telles que des serveurs, des réseaux, du stockage de données, des services et des applications, qui peuvent être rapidement approvisionnés et libérés avec un minimum d'efforts de gestion ou d'interaction avec le fournisseur de services. Avec les progrès de la technologie, les coûts d'hébergement d'applications, de calcul, de stockage et de livraison de données ont considérablement diminué.

Le cloud computing permet la fourniture de services informatiques via Internet. Il permet aux entreprises et aux particuliers d'utiliser du matériel et des logiciels gérés par des tiers à distance. Le cloud computing permet d'accéder aux ressources informatiques et aux informations depuis n'importe où, à tout moment, via Internet.

Les utilisateurs du cloud n'ont pas besoin de connaissances ou d'expertise sur l'infrastructure technologique.

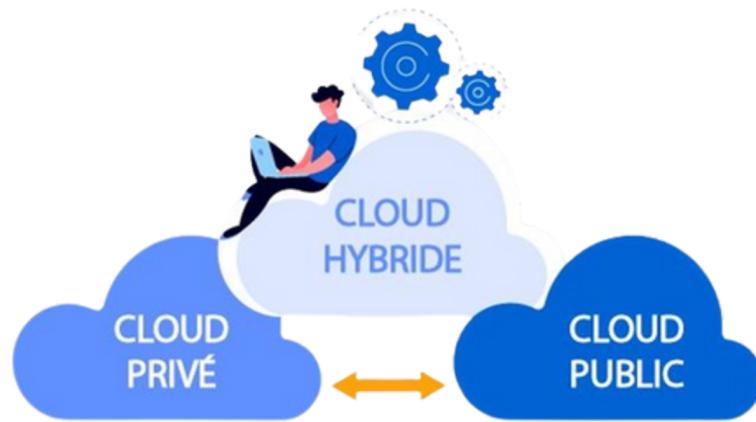
Les principaux attributs du cloud sont :

-  Expérience utilisateur améliorée
-  Mise à l'échelle élastique
-  Approvisionnement automatisé
-  Très virtualisé



2.1 Les Environnements de Cloud

- Modèles de déploiement du cloud :



- **Cloud public** : Les ressources sont fournies via Internet par un fournisseur tiers et sont partagées entre plusieurs clients.
- **Cloud privé** : L'infrastructure cloud est exploitée exclusivement pour une seule organisation.
- **Cloud hybride** : Combinaison du cloud public et du cloud privé, permettant une plus grande flexibilité.
- **Cloud communautaire** : L'infrastructure est partagée entre plusieurs organisations ayant des intérêts communs.

Remarque:

Concernant tous ces modèles de cloud, ils sont basés sur une infrastructure (data center) où l'on peut trouver des serveurs, systèmes de stockage, équipements réseau .

Par exemple, pour le cloud hybride, un individu ou une organisation qui dispose d'un cloud privé (leur propre infrastructure) peut vouloir étendre ou même utiliser d'autres serveurs. Dans ce cas, ils se tourneront vers le cloud public.

- La différence entre les modèles de déploiement du cloud :



Cloud public

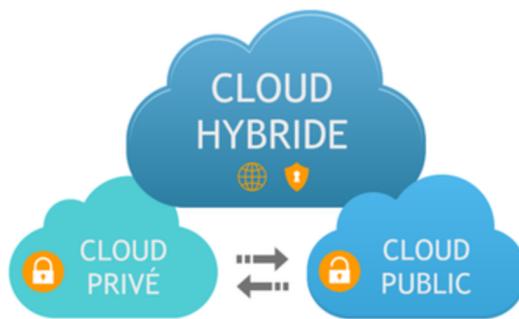
- Les ressources (serveurs, stockage, etc.) appartiennent et sont gérées par un fournisseur cloud externe (Amazon, Microsoft, Google, etc.).
 - Accessible à tous via Internet, avec des modèles de tarification à l'utilisation.
 - Niveau de sécurité et de confidentialité généralement plus faible que le cloud privé.
 - Avantages : évolutivité, flexibilité, faible coût d'entrée.
 - Inconvénients : moins de contrôle, préoccupations de sécurité et de confidentialité.
-

Cloud privé

- Les ressources appartiennent et sont contrôlées par l'organisation elle-même.
- Accès restreint uniquement aux utilisateurs autorisés au sein de l'organisation.
- Niveau de sécurité et de confidentialité plus élevé que le cloud public.
- Avantages : contrôle total, sécurité renforcée.
- Inconvénients : coûts d'investissement plus élevés, nécessite une expertise interne.

Cloud hybride

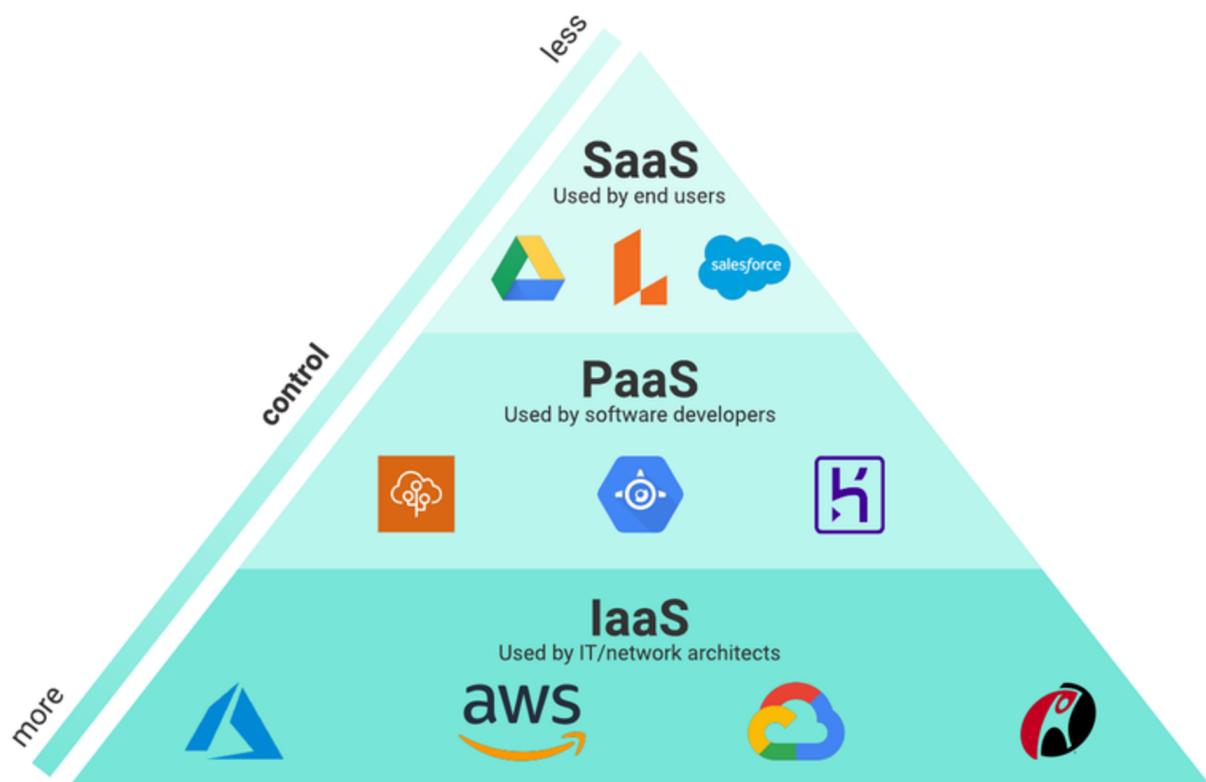
- Combinaison du cloud privé et du cloud public.
- Permet d'étendre les capacités du cloud privé avec des ressources cloud publiques.
- Offre une plus grande flexibilité et évolutivité tout en maintenant un niveau de contrôle et de sécurité élevé.
- Avantages : flexibilité, évolutivité, sécurité renforcée.
- Inconvénients : complexité de gestion, intégration entre les environnement .



Cloud communautaire

- Partagé entre plusieurs organisations ayant des objectifs, des exigences ou des préoccupations communes.
- Offre les avantages d'un cloud privé tout en bénéficiant d'économies d'échelle.
- Niveau de sécurité et de confidentialité plus élevé que le cloud public.
- Avantages : économies d'échelle, sécurité renforcée.
- Inconvénients : dépendance à la communauté, complexité de gestion.

- Principaux services cloud



Infrastructure en tant que service (IaaS)

→ Fournit un accès à la demande à des ressources de calcul, de stockage et de réseau.

Plateforme en tant que service (PaaS)

→ Offre une plateforme de développement et de déploiement d'applications.

Logiciel en tant que service (SaaS)

→ Permet l'accès à des applications hébergées en ligne par le fournisseur.

2.2 Avantages du Cloud Computing

Rentable

Le cloud computing est probablement le moyen le plus rentable d'utiliser, de maintenir et de mettre à jour des applications. Les logiciels de bureau traditionnels coûtent cher. En additionnant les frais de licence pour plusieurs utilisateurs dans une entreprise, cela peut s'avérer très coûteux.

Stockage illimité

Stocker des informations dans le cloud offre une capacité de stockage pratiquement illimitée.

Sauvegarde et récupération

Étant donné que toutes les données sont stockées sur le cloud, les sauvegarder et les restaurer est relativement plus simple que de stocker les données sur un appareil physique. L'ensemble du processus de la sauvegarde et la restauration deviennent beaucoup plus simples que les autres méthodes traditionnelles.

Intégration logicielle automatique

L'intégration logicielle est généralement quelque chose qui se produit automatiquement sur le nuage. Cela signifie que les utilisateurs du cloud n'ont pas à faire d'efforts supplémentaires pour personnaliser et intégrer leurs applications.

Accès facile aux informations

Une fois les utilisateurs inscrits sur le cloud, ils peuvent accéder à leurs informations depuis n'importe où via une connexion Internet.

Déploiement rapide

Le cloud computing offre l'avantage d'un déploiement rapide. L'ensemble du système peut être entièrement fonctionnel en quelques minutes.

2.3 Inconvénient du Cloud Computing

Manque de contrôle

Avec le cloud, l'utilisateur aura moins de contrôle sur votre infrastructure et vos données que si vous les hébergez vous-même.

Sécurité sur le cloud

Le stockage d'informations dans le cloud peut rendre les entreprises vulnérables aux attaques et menaces provenant de l'extérieur, notamment : Accès non autorisé, Violation de la confidentialité ,
Perte de contrôle...

Vendor Lock-In

Une fois qu'un utilisateur a choisi un fournisseur de cloud, il peut être difficile de migrer vers une autre plateforme en raison des coûts et de la complexité impliqués.

Latence et performances

Selon l'emplacement géographique des utilisateurs et des serveurs cloud, ils peuvent rencontrer des problèmes de latence qui affectent les performances de leurs applications.

Dépendance à la connexion Internet

Avec le cloud, vous dépendez entièrement d'une connexion Internet stable et fiable. En cas de panne ou de problème de réseau, vous pouvez être confronté à une indisponibilité de vos données et applications.

Problèmes de conformité et de réglementation

Selon le secteur d'activité, il peut être difficile de garantir le respect des réglementations en matière de confidentialité et de sécurité des données avec le cloud.

3.0 Qu'est que c'est le concept de virtualisation

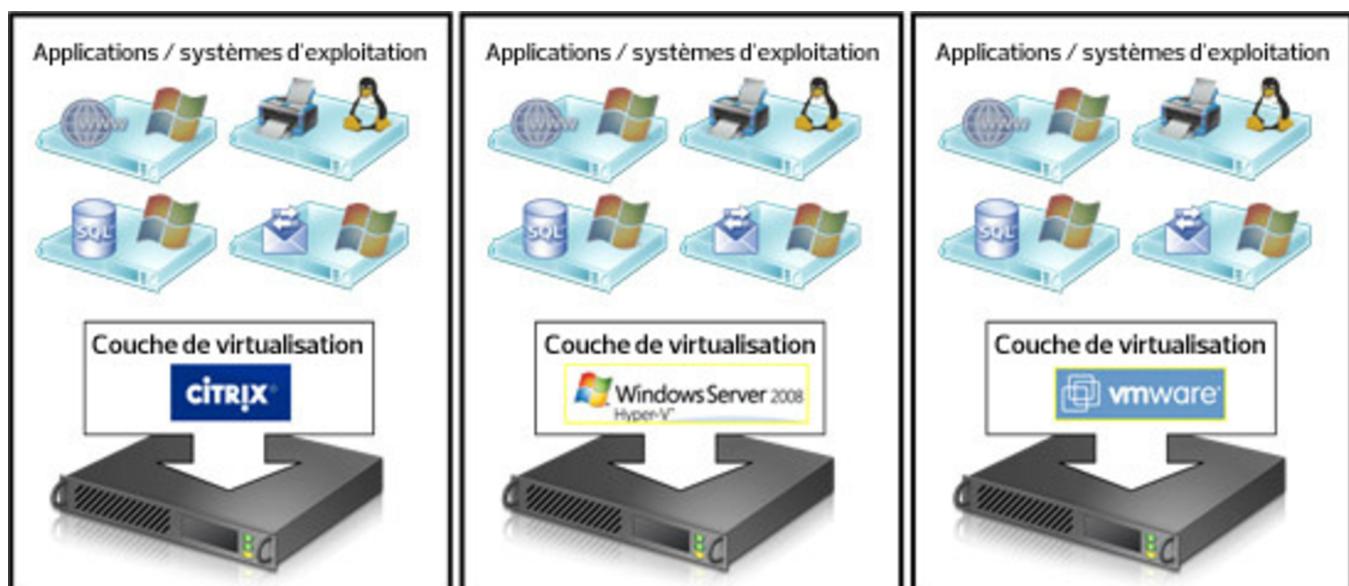
3.1 Avant la virtualisation

 "Une application sur un seul serveur" était le problème principal pour les entreprises lorsqu'il s'agissait de déployer des applications sur des serveurs, car cela entraînait une consommation élevée de ressources pour un simple déploiement.

Par exemple, imaginons que nous ayons 3 applications à déployer : un site Web, une base de données et un service de messagerie.

Le déploiement en soi n'était pas un problème majeur, mais les ressources nécessaires pour ce déploiement étaient trop importantes et nous coûtaient beaucoup d'argent.

Pour les grandes entreprises, ce n'était pas un problème, mais lorsque le nombre d'applications augmentait, cela entraînait une baisse financière due à l'achat de nouvelles machines pour lancer des serveurs de déploiement, sachant que le déploiement d'une application nécessitait un serveur dédié (une machine) qui ne servirait que pour cette application, car les systèmes d'exploitation comme Linux et Windows n'avaient pas la capacité d'exécuter plusieurs applications de manière sécurisée sur un seul serveur.



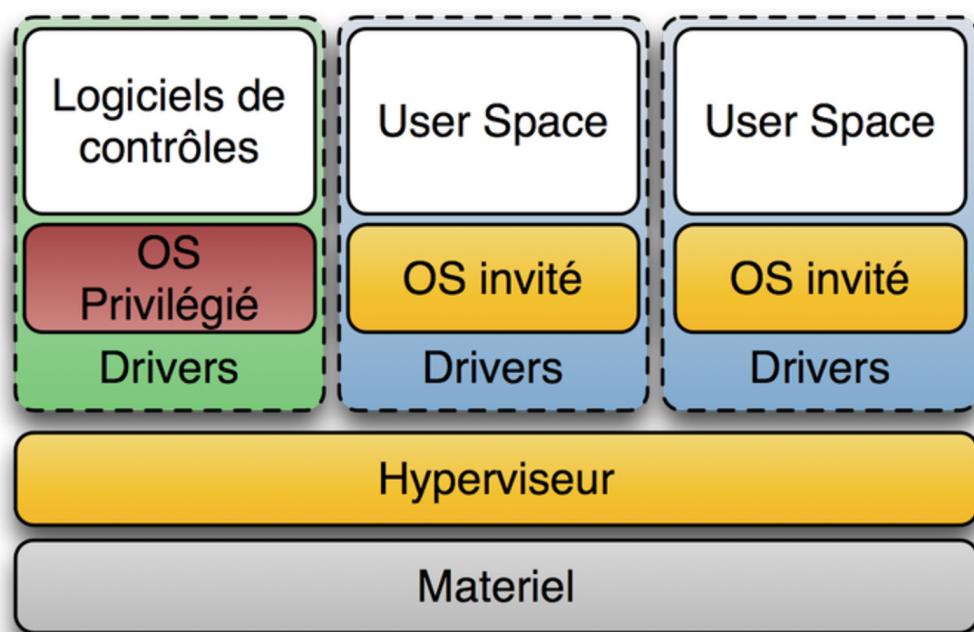
C'est ainsi qu'est apparue la nécessité de trouver une solution à ce problème : "une application va coûter un serveur" qui gaspillait de l'argent et n'exploitait pas pleinement les capacités des serveurs modernes, souvent sous-utilisés.

Et c'est alors que la solution des machines virtuelles (VM) est apparue, aidant à réduire cette perte. Cependant, ce n'était toujours pas la solution idéale à notre problème principal.

3.2 Définition

La virtualisation est une technologie qui permet de créer et de gérer des ressources informatiques virtuelles, telles que des serveurs, des réseaux et des infrastructures, en utilisant les ressources physiques comme le processeur, la mémoire, le stockage, etc.

Grâce à la virtualisation, il est possible d'exécuter plusieurs machines virtuelles (VM) indépendantes sur une seule machine physique, chacune avec son propre système d'exploitation, ses applications et ses configurations. Cela permet de mieux exploiter la capacité des serveurs physiques en les partageant entre plusieurs charges de travail, réduisant ainsi les coûts et améliorant l'agilité informatique.



On peut pas parler du virtualisation sans citer ces principaux composants :

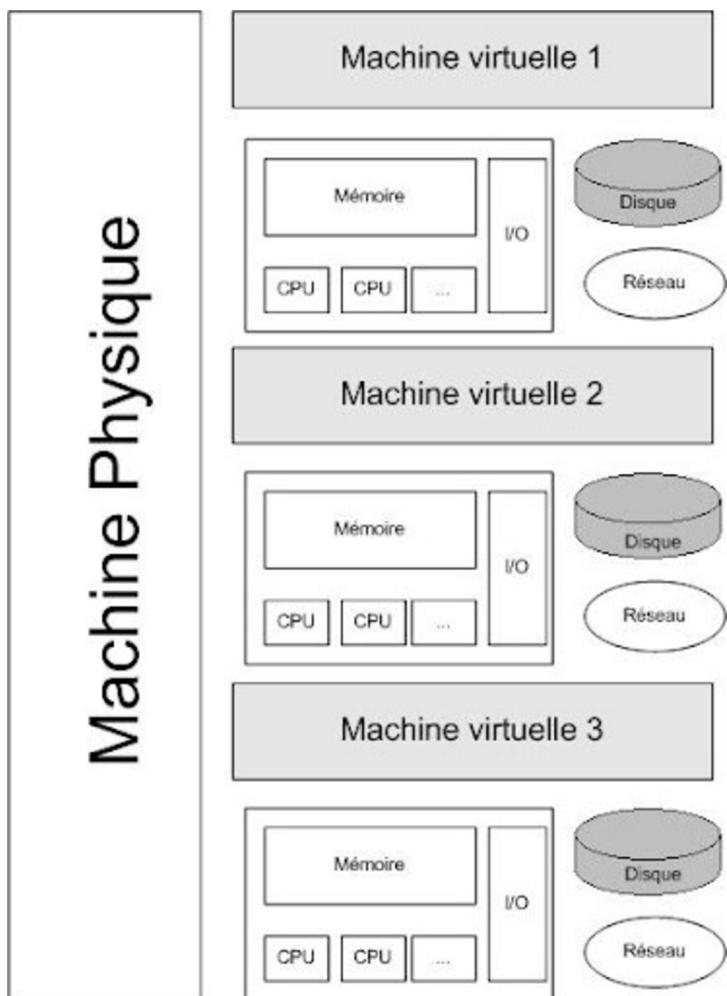
- **Hyperviseur (Hypervisor)** : C'est le logiciel central qui permet de créer et de gérer les machines virtuelles.
- **Machines virtuelles (Virtual Machines - VMs)** : Ce sont des environnements d'exécution logiciels émulant un système d'exploitation complet.
Chaque VM a son propre OS, applications, configurations, indépendamment des autres.
- **Système invité**: Les machines virtuelles (VMs) ou les conteneurs hébergent leurs propres systèmes d'exploitation, applications et configurations, isolés les uns des autres.

Constatation

En se basant sur la figure à droite
on constate que la virtualisation est une
technologie qui nous permet de :

- Créez plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique.

Machine Physique



3.3 Petite histoire sur la virtualisation

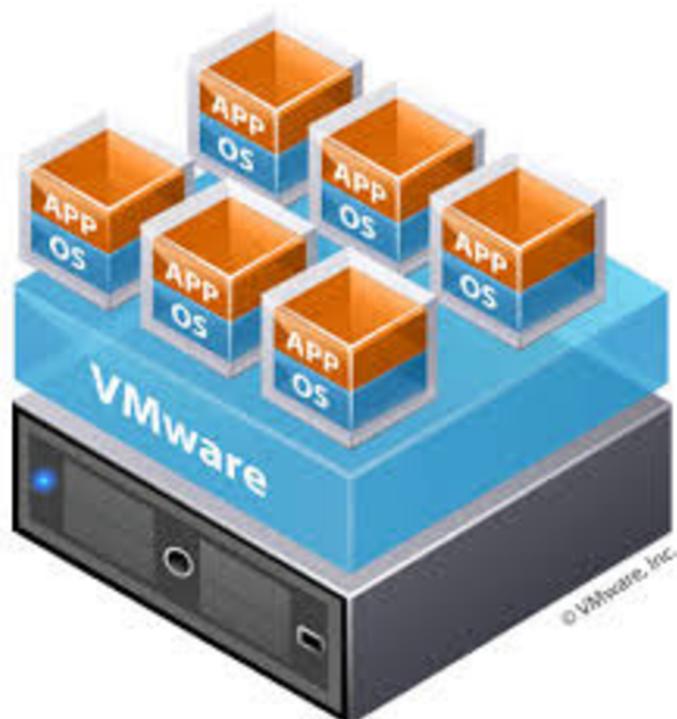
Bien que le concept de virtualisation soit apparu dans les années 1960, ce n'est que dans les années 2000 que son utilisation s'est généralisée.

Les technologies qui sont à la base de la virtualisation, telles que les hyperviseurs, ont été développées il y a plusieurs dizaines d'années afin de permettre à divers utilisateurs d'accéder simultanément aux ordinateurs qui effectuaient des traitements par lots.

Cependant, d'autres solutions comme le temps partagé ont gagné en popularité, reléguant la virtualisation au second plan pendant longtemps.

Ce n'est qu'au cours des années 1990 que la virtualisation a pris son envol, en réponse aux problèmes posés par les serveurs physiques et les piles informatiques propriétaires.

La virtualisation a permis aux entreprises de partitionner leurs serveurs et d'exécuter les applications existantes sur plusieurs types et versions de systèmes d'exploitation, optimisant ainsi l'utilisation de leurs infrastructures et réduisant les coûts.

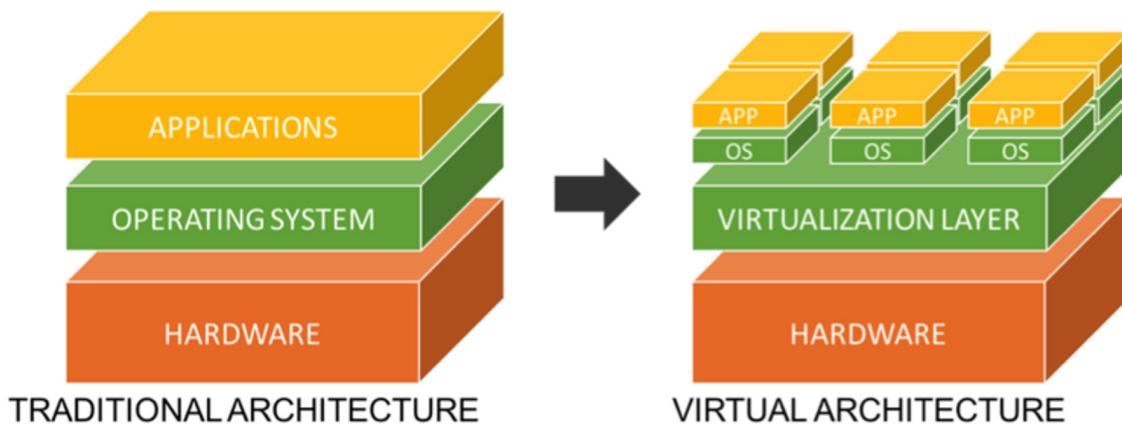


3.4 Fonctionnement



Le processus de virtualisation dans l'architecture est différent de l'architecture classique. Comme on le voit dans la figure, dans l'architecture classique, il y a seulement la couche matérielle, la couche système d'exploitation (OS) et la couche applicative.

Tandis que pour l'architecture de virtualisation, on trouve l'hyperviseur au-dessus de la couche matérielle. L'hyperviseur joue le rôle d'un logiciel avec pour but de gérer les différents environnements virtuels.



Cela nous amène à penser à l'importance de l'hyperviseur, car il semble que tout le concept de virtualisation soit basé sur lui.

- **Hyperviseur**

Un hyperviseur est un logiciel qui permet la virtualisation des ressources informatiques. Il est responsable de la création, de l'exécution et de la gestion des machines virtuelles sur un système physique.

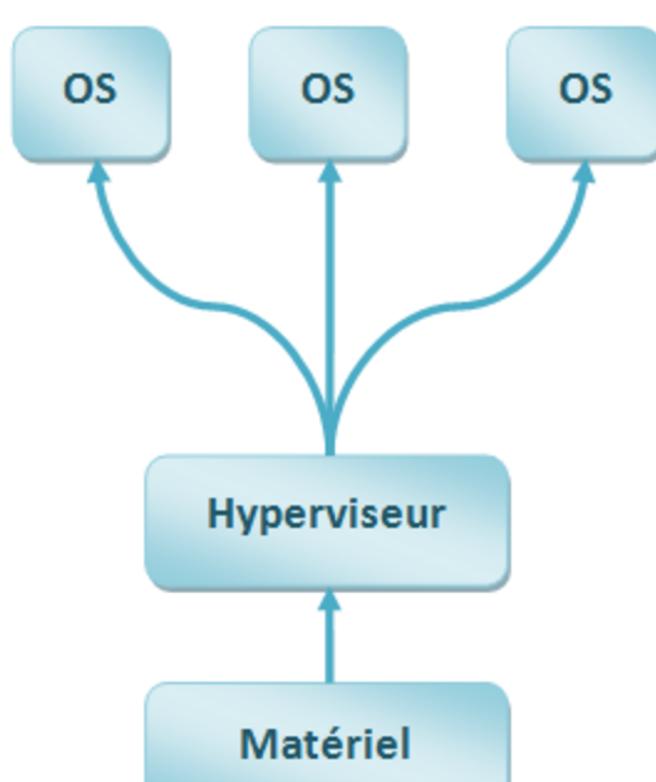
L'hyperviseur isole le système d'exploitation et les ressources des machines virtuelles, permettant ainsi à plusieurs systèmes d'exploitation de fonctionner en parallèle sur le même matériel physique.

Hyperviseurs de type 1 :

Les hyperviseurs de type 1, également appelés "hyperviseurs natif", s'installent directement sur le matériel physique, sans système d'exploitation intermédiaire.

Ils interagissent donc directement avec la couche matériels (le processeur, la mémoire et le stockage de la machine hôte).

- **VMware ESXi et la suite vSphere** sont également basés sur un noyau Linux.
- **Proxmox VE** (Linux KVM) également.
- **Microsoft Hyper-V** : lorsqu'on installe Hyper-V, il se positionne en dessous de Windows. Donc le système Windows est modifié de manière à être une couche au-dessus.
Malgré son mode d'installation qui peut paraître un peu particulier et qui fait plutôt penser à un hyperviseur de type 2.

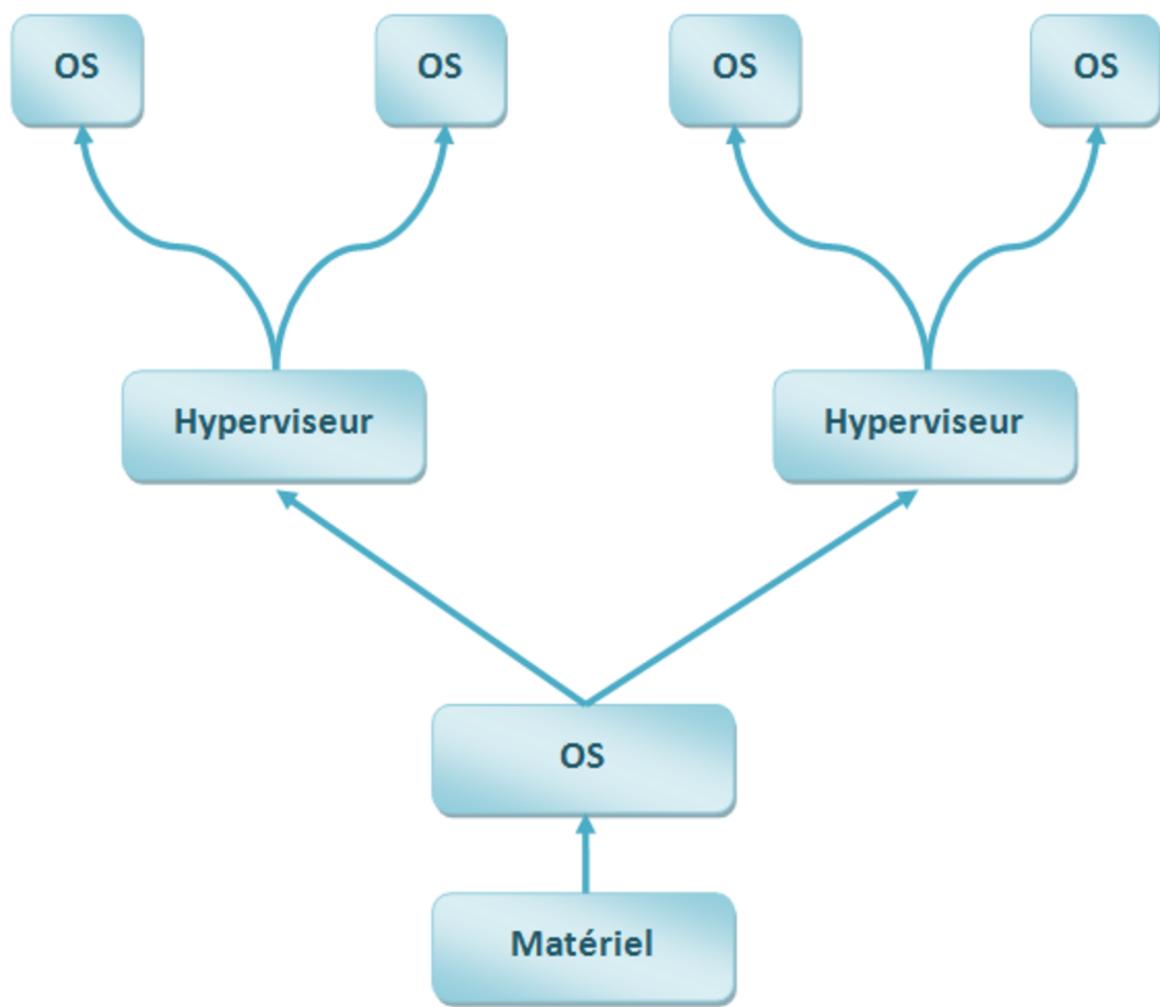


Hyperviseurs de type 2 :

Les hyperviseurs de type 1, également appelés "hyperviseurs natif", s'installent directement sur le matériel physique, sans système d'exploitation intermédiaire.

Ils interagissent donc directement avec la couche matériels (le processeur, la mémoire et le stockage de la machine hôte).

- **Oracle VirtualBox**
- **Vmware Workstation (payante)**
- **Vmware Workstation Player (gratuite)**
- **Vmware Fusion pour MacOs**



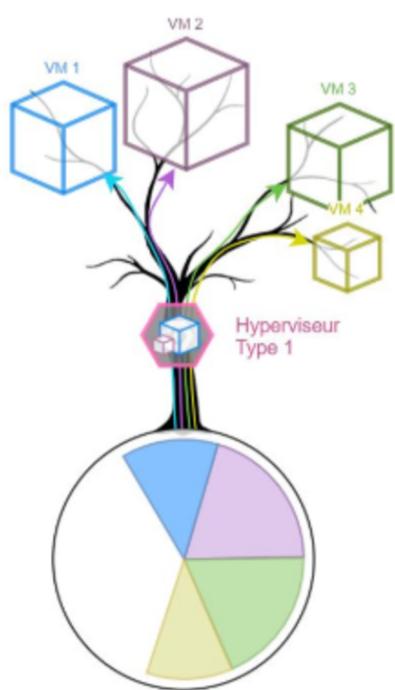
• Difference entre les types d'hyperviseurs:

Couche d'exécution

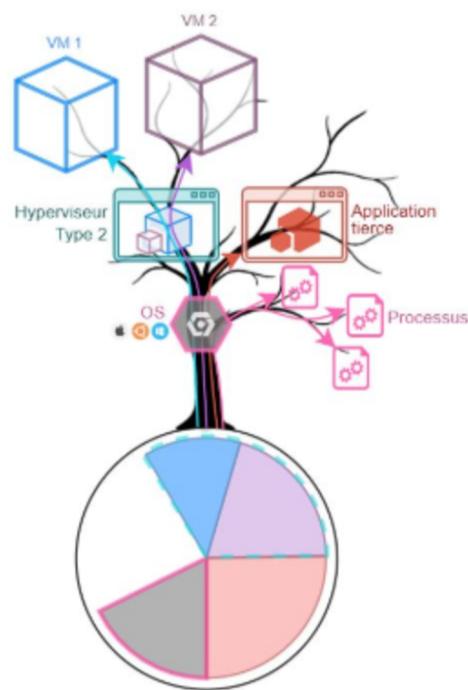
- Les hyperviseurs de **type 1** s'exécutent directement sur le matériel physique, sans système d'exploitation intermédiaire.
- Les hyperviseurs de **type 2** fonctionnent comme une application sur un système d'exploitation préexistant.

Allocation des ressources

- Les hyperviseurs de **type 1** ont un accès direct aux ressources matérielles et peuvent implémenter leurs propres stratégies d'allocation.
- Les hyperviseurs de **type 2** doivent négocier l'accès aux ressources avec le système d'exploitation hôte, ce qui réduit leur efficacité.



Fonctionnement d'un hyperviseur de type 1



Fonctionnement d'un hyperviseur de type 2



Facilité d'installation et d'utilisation

- Les hyperviseurs de **type 1** nécessitent des compétences de niveau administrateur système pour leur configuration et gestion.
- Les hyperviseurs de **type 2** sont plus faciles à installer et à gérer car ils fonctionnent comme une application sur un système d'exploitation.



Performances

- Les hyperviseurs de **type 1** offrent de meilleures performances pour les machines virtuelles car ils n'ont pas besoin de négocier les ressources.
- Les performances des hyperviseurs de **type 2** sont réduites car ils ne peuvent utiliser que les ressources fournies par le système d'exploitation hôte.



Sécurité

- Les hyperviseurs de **type 1** présentent moins de risques de sécurité car ils sont le seul logiciel entre le matériel et les machines virtuelles.
- Les hyperviseurs de **type 2** dépendent du système d'exploitation hôte et peuvent être plus exposés aux activités malveillantes.

• Fonctionnement de la virtualisation

Concernant le fonctionnement de la virtualisation, tout est relié à l'hyperviseur. C'est pourquoi j'ai inclus la partie hyperviseur pour bien comprendre ses bases, ses types, etc.

Le caractère le plus important qui fait le lien entre la virtualisation et l'hyperviseur est que l'hyperviseur répartit les ressources physiques pour permettre aux environnements virtuels de les utiliser.

Ces ressources sont partitionnées à partir de l'environnement physique et distribuées aux différents environnements virtuels.

Les utilisateurs interagissent avec ces environnements (également appelés machines virtuelles ou hôtes) et y exécutent des calculs.

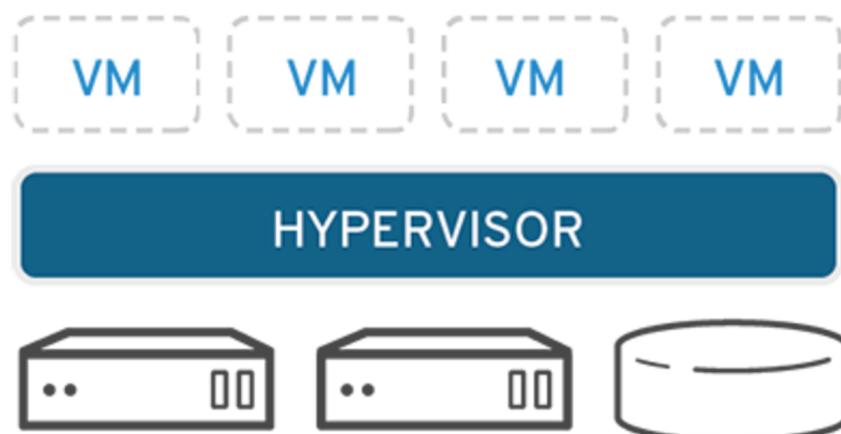
La machine virtuelle opère comme un fichier de données unique. Comme n'importe quel fichier numérique, vous pouvez la transférer d'un ordinateur à un autre, l'ouvrir sur l'un ou l'autre et l'utiliser de la même manière.

Lorsque l'environnement virtuel est exécuté et qu'un utilisateur ou un programme émet une instruction nécessitant des ressources supplémentaires à partir de l'environnement physique, l'hyperviseur transmet cette requête au système physique et met en cache les modifications.

3.5 Types de ressources virtualisées

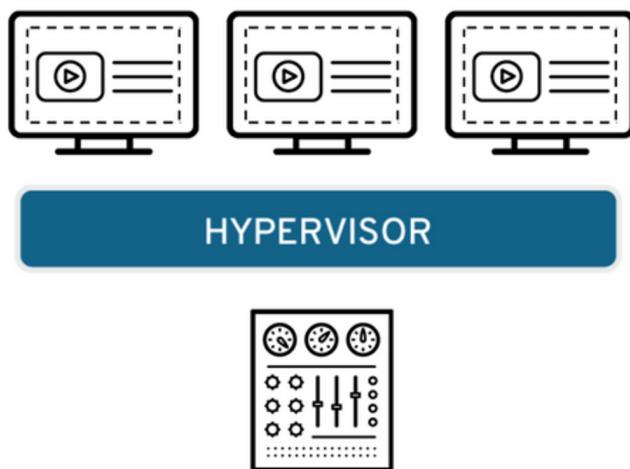
Virtualisation des serveurs

- Permet de créer plusieurs machines virtuelles sur un même serveur physique.
- Chaque machine virtuelle peut exécuter son propre système d'exploitation et applications.



Virtualisation des postes de travail

- Permet de déployer des environnements de bureau virtuels accessibles depuis différents appareils.
- Facilite le déploiement et la gestion des applications.



3.6 Machines virtuelles

• **Definition**

Les machines virtuelles permettent d'exécuter plusieurs applications sur un seul serveur en simulant le matériel et le logiciel.

Par exemple, reprenons le cas précédent des 3 applications. Avec les machines virtuelles, ces 3 applications peuvent être exécutées sur un seul serveur en simulant ces 3 serveurs et leurs applications en créant 3 machines virtuelles. Ainsi, ce seul serveur exécute trois machines virtuelles ou logicielles.

Il exécute tous les différents systèmes d'exploitation ainsi que les applications (site web, base de données, e-mail) côte à côte sur une seule machine.

• **Avantages**

Flexibilité : La création d'une machine virtuelle est plus rapide et plus facile que l'installation d'un système d'exploitation sur un serveur physique.

Il est possible de cloner une machine virtuelle avec le système d'exploitation déjà installé.

Meilleur utilisation des ressources : Comme la plupart des machines virtuelles fonctionnent sur un seul ordinateur physique, il n'est pas nécessaire d'acheter un nouveau serveur pour faire fonctionner un autre système d'exploitation

Évolutivité: Grâce au cloud computing, il est devenu plus facile d'introduire plusieurs copies d'une même machine virtuelle pour mieux gérer des charges de travail accrues.

• **Lien entre les machines virtuelles et le cloud**

1. Rôle central des machines virtuelles dans le cloud

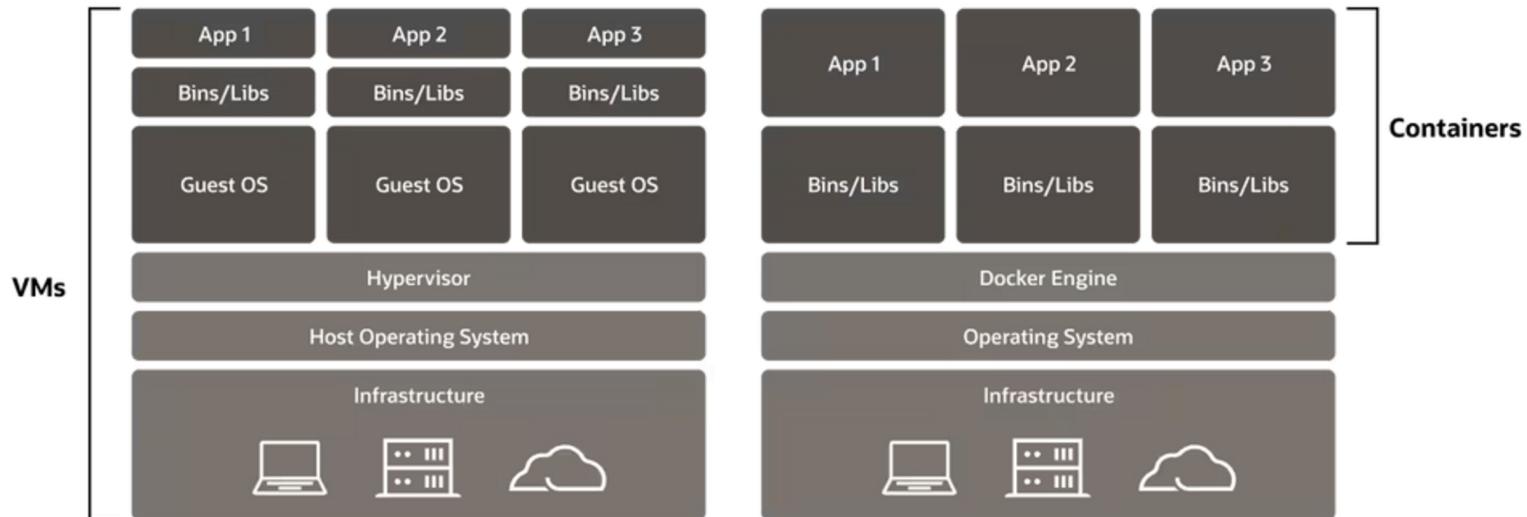
- Les machines virtuelles (VM) constituent l'unité de calcul fondamentale dans les environnements de cloud computing.
- Les fournisseurs de cloud utilisent massivement les VM pour permettre l'exécution et la mise à l'échelle des applications et des charges de travail dans le cloud.

2. Virtualisation et cloud computing

- La virtualisation permet de découpler les ressources informatiques du matériel physique, facilitant ainsi la mise en place d'environnements cloud.
- Le découplage se base sur l'hyperviseur, ressource partagées et l'isolation .

3.7 Conteneurs

- Un conteneur est un format d'empaquetage qui regroupe tout le code et les dépendances d'une application dans un format standard, qui permet une exécution rapide et de fiable dans l'ensemble des environnements informatiques.



Virtual Machines

- Each virtual machine (VM) includes the app, the necessary binaries and libraries and an entire guest operating system

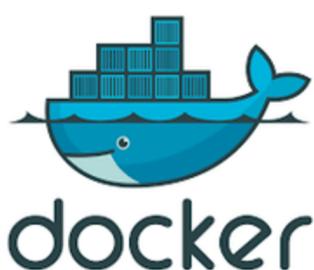
Containers

- Containers include the app and all of its dependencies, but share the kernel with other containers.
- Run as an isolated process in userspace on the host operating system.
- Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud.

• Qu'est-ce que Docker:

Docker est un conteneur exécutable populaire léger et autonome, qui comprend tous les éléments nécessaires pour exécuter une application, notamment les bibliothèques, les outils système, le code et le runtime.

Docker est également une plateforme logicielle qui permet aux développeurs de créer, tester et déployer rapidement des applications en conteneur.



- **Moteur Docker :**

Ce logiciel hôte open source crée et exécute les conteneurs. Le moteur Docker fait office d'application client-serveur prenant en charge les conteneurs sur divers serveurs Windows et systèmes d'exploitation Linux, notamment Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE et Ubuntu.

- **Images Docker :**

Ensemble de logiciels à exécuter en tant que conteneur contenant un ensemble d'instructions pour la création d'un conteneur pouvant être exécuté sur la plate-forme Docker. Les images ne sont pas modifiables ; pour apporter des modifications à une image, il faut en créer une nouvelle.

- **Registre Docker:**

Emplacement pour stocker et télécharger des images. Le registre est une application côté serveur sans conservation de statut et évolutive qui stocke et distribue des images Docker.

- **Les services Containers as a Service (CaaS) :**

Les services Containers as a Service (CaaS) ou services de conteneurs sont des services cloud gérés qui gèrent le cycle de vie des conteneurs. Les services de conteneur aident à orchestrer (démarrer, arrêter, mettre à l'échelle) l'exécution des conteneurs. Grâce aux services de conteneur, vous pouvez simplifier, automatiser et accélérer le développement de vos applications et le cycle de vie de votre déploiement.



Azure Kubernetes Service (AKS)



- **Docker et les services de conteneurs**

Docker et les services de conteneurs ont connu une adoption rapide et un énorme succès au cours des dernières années. Alors qu'il s'agissait d'une technologie open source presque inconnue et plutôt technique en 2013, Docker a évolué pour devenir un environnement d'exécution standardisé désormais officiellement pris en charge pour de nombreux produits Oracle Enterprise.



3.8 Kubernetes

kubernetes

Kubernetes, souvent abrégé en K8s, est une plateforme open source conçue pour automatiser la gestion, l'orchestration et le déploiement de conteneurs d'application à grande échelle. Développé initialement par Google et rendu open source en 2014, Kubernetes s'inspire des pratiques de gestion de conteneurs utilisées dans les services de Google, comme Gmail et Google Maps, qui génèrent un volume immense de conteneurs chaque semaine

- **Fonctionnalités et Utilité**

Orchestration des Conteneurs : Il permet de gérer des applications composées de plusieurs conteneurs, en assurant leur déploiement, leur mise à l'échelle et leur gestion au fil du temps.

Abstraction de l'Infrastructure : Les développeurs peuvent se concentrer sur le développement d'applications sans se soucier des détails de l'infrastructure sous-jacente, car Kubernetes gère les ressources nécessaires.

Surveillance et Récupération : Kubernetes surveille l'état des conteneurs et redémarre ceux qui échouent, garantissant ainsi la disponibilité des services.

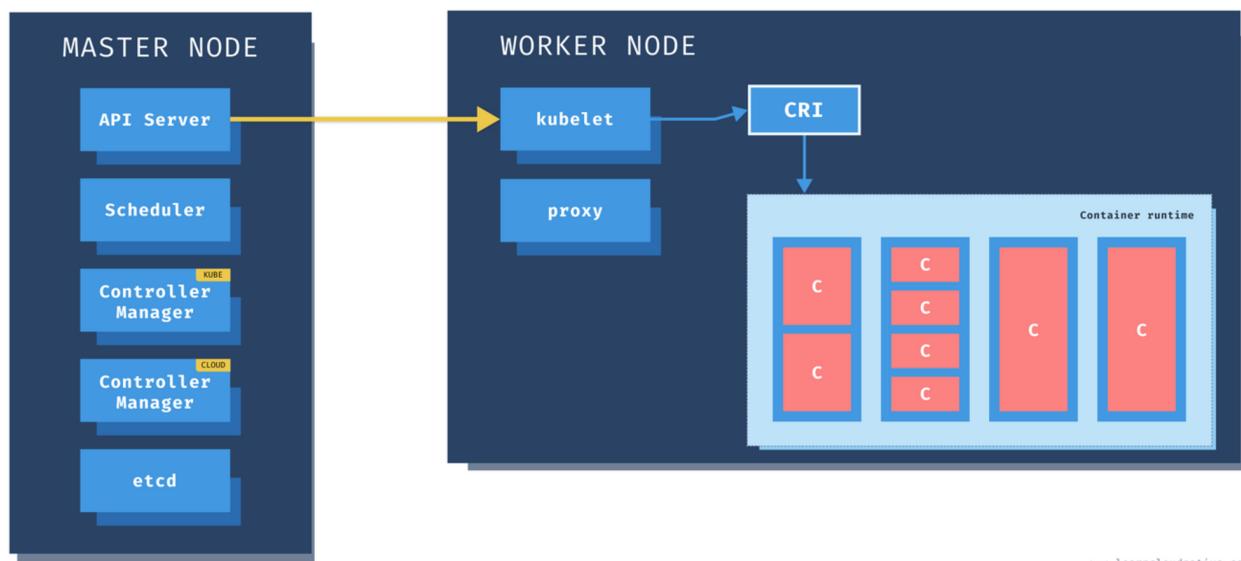
Extensibilité et Portabilité : Grâce à son architecture modulaire, Kubernetes peut être déployé sur divers environnements, que ce soit sur des serveurs physiques, des machines virtuelles ou dans le cloud, tout en maintenant la portabilité entre différents fournisseurs de services cloud.

• L'architecture du Kubernetes

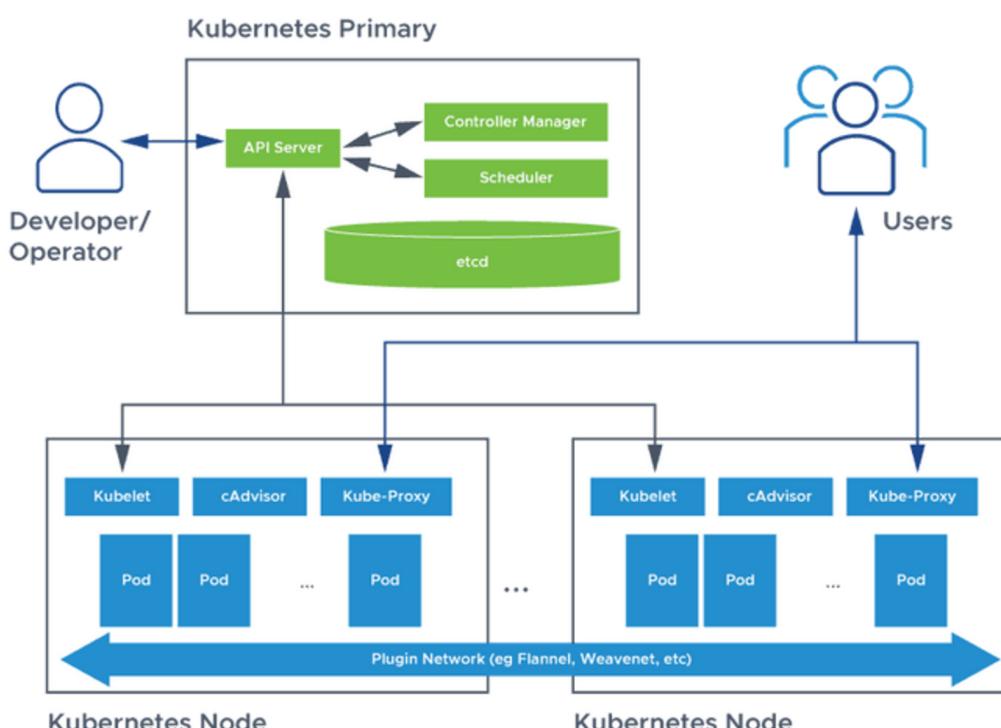
Un cluster Kubernetes est un ensemble de machines physiques ou virtuelles et d'autres ressources d'infrastructure nécessaires pour exécuter vos applications conteneurisées.

Chaque machine dans un cluster Kubernetes est appelée un nœud. Il existe deux types de nœuds dans chaque cluster Kubernetes :

- **Nœud maître** : ce nœud héberge le plan de contrôle Kubernetes et gère le cluster.
- **Nœud de travail** : exécute vos applications conteneurisées.



www.learncloudnative.com



- **Nœud maître :**

Le nœud maître est responsable de la gestion de l'état global du cluster. Il prend des décisions sur la planification des pods et surveille l'état des nœuds et des applications.

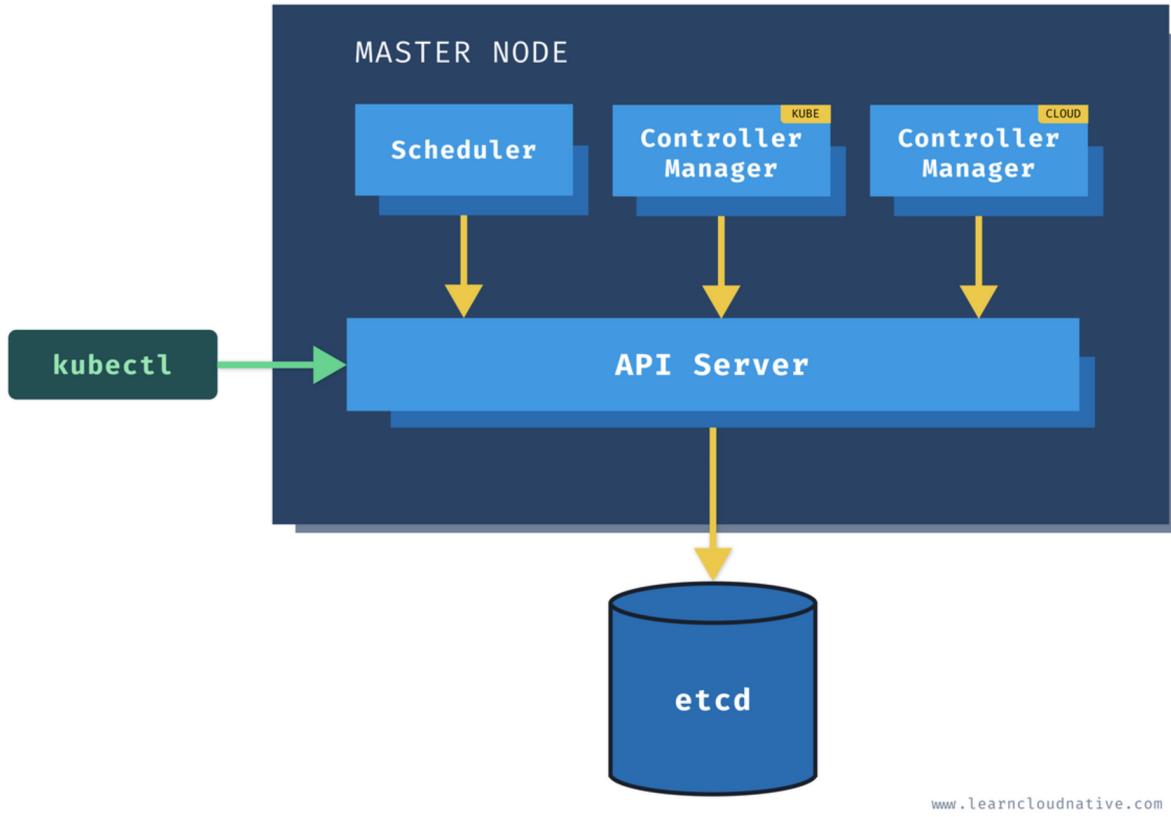
Les principaux composants du nœud de contrôle incluent :

- **kube-apiserver** : L'un des composants principaux du nœud maître est appelé le serveur API. Le serveur API est le point de terminaison avec lequel la ligne de commande Kubernetes (kubectl) communique lorsque vous créez des ressources Kubernetes ou gérez le cluster.
- **etcd** : Une base de données clé-valeur qui stocke l'état du cluster.
- **kube-scheduler** : Assigne les pods aux nœuds en fonction de la disponibilité des ressources.
- **kube-controller-manager** : Gère les contrôleurs qui assurent que l'état du cluster correspond à la configuration souhaitée.

Il existe deux types de controller-manager qui s'exécutent sur les nœuds maîtres.

Le gestionnaire de contrôleurs kube : Ces contrôleurs surveillent l'état du cluster et essaient de réconcilier l'état actuel du cluster.

Le gestionnaire de contrôleurs cloud: Exécute des contrôleurs spécifiques au fournisseur de cloud et peut gérer des ressources en dehors de votre cluster.



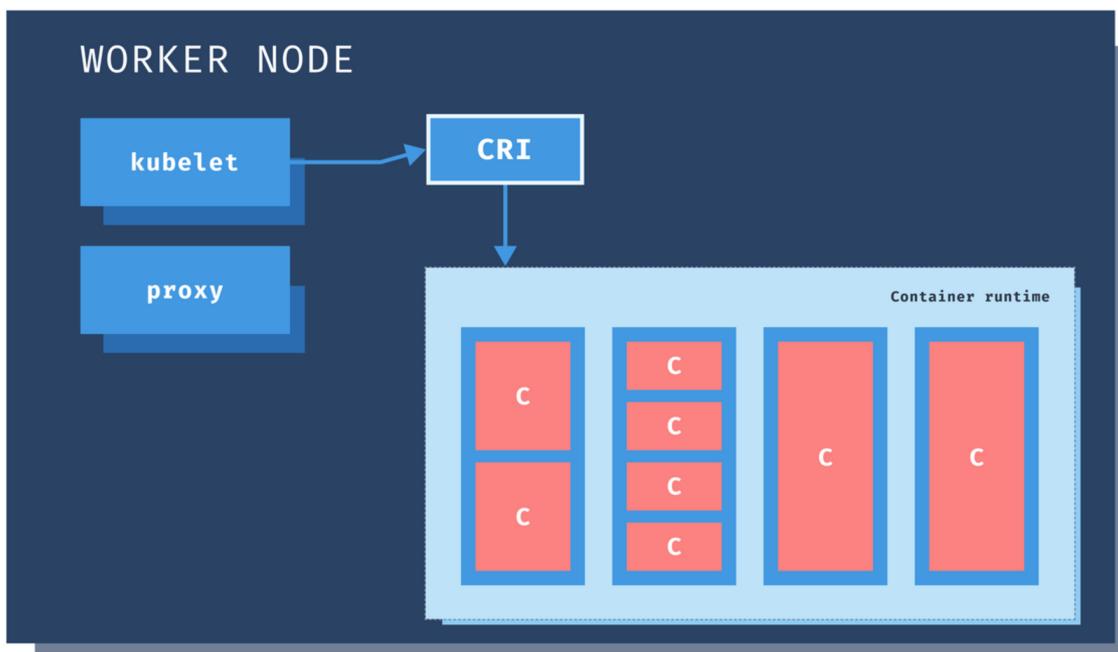
www.learncloudnative.com

Nœud maître :

Tout comme sur le nœud maître, les nœuds de travail ont également différents composants qui s'exécutent. Le premier est le **kubelet**. Ce service s'exécute sur chaque nœud de travail et sa tâche est de gérer les conteneurs. Il s'assure que les conteneurs sont en cours d'exécution et en bonne santé, et il se connecte au plan de contrôle. Le kubelet communique avec le serveur API et est responsable de la gestion des ressources sur le nœud sur lequel il s'exécute.

Lorsqu'un nouveau nœud de travail est ajouté au cluster, le kubelet se présente et fournit les ressources qu'il possède (par exemple, "J'ai X CPU et Y mémoire"). Ensuite, il demande si des conteneurs doivent être exécutés. Vous pouvez considérer le kubelet comme un gestionnaire de nœud de travail.

Kubelet uses the **container runtime interface** (CRI) to talk to the container runtime. The container runtime is responsible for working with the containers.



www.learncloudnative.com

Les conteneurs s'exécutent à l'intérieur de pods, représentés par les rectangles bleus dans la figure ci-dessus (les conteneurs sont les rectangles rouges à l'intérieur de chaque pod).

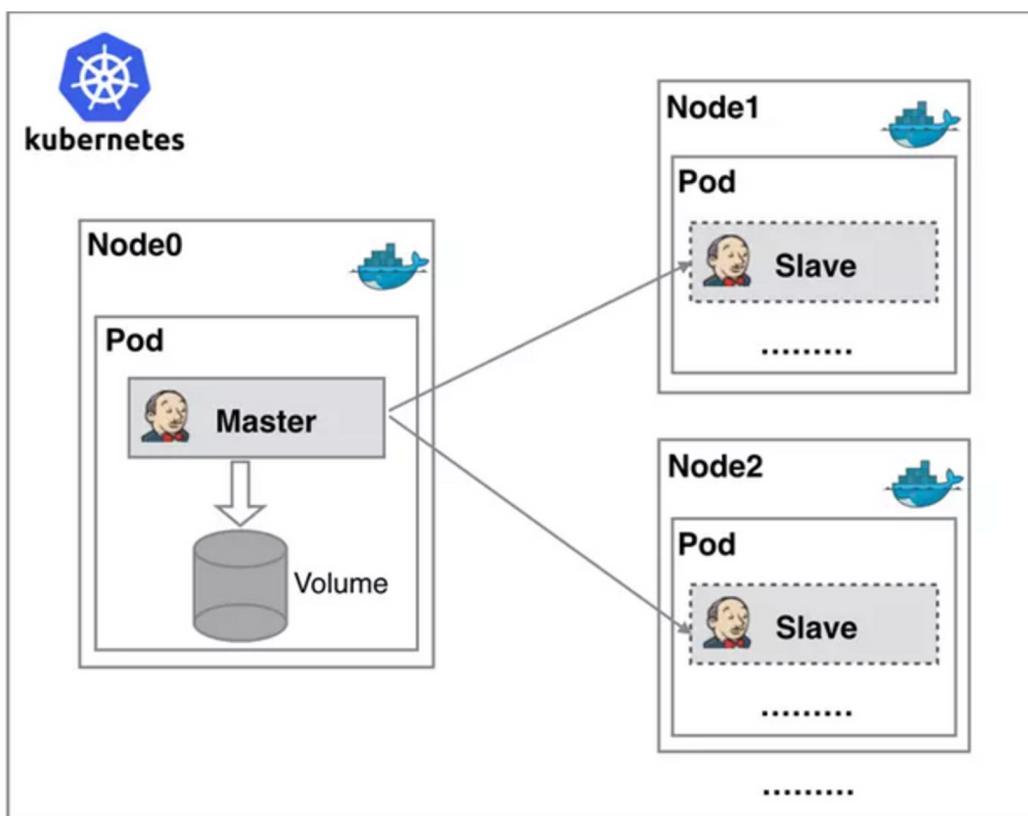
Un pod est la plus petite unité déployable qui peut être créée, planifiée et gérée sur un cluster Kubernetes. Un pod est une collection logique de conteneurs qui composent votre application. Les conteneurs exécutés à l'intérieur du même pod partagent également le réseau et l'espace de stockage.

Chaque nœud de travail dispose également d'un **proxy** qui agit en tant que proxy réseau et équilibrEUR de charge pour les charges de travail exécutées sur les nœuds de travail. Les requêtes des clients qui passent par un équilibrEUR de charge externe sont redirigées vers les conteneurs exécutés à l'intérieur du pod via ces proxys.

- **Les principaux concepts liés à Kubernetes**

Noeud

Un nœud est une machine (physique ou virtuelle) qui exécute des pods. Chaque nœud dans un cluster Kubernetes contient les services nécessaires pour faire fonctionner les pods, comme le kubelet et le runtime de conteneur. Les nœuds sont des ressources de calcul qui hébergent les applications déployées sous forme de pods



Pod

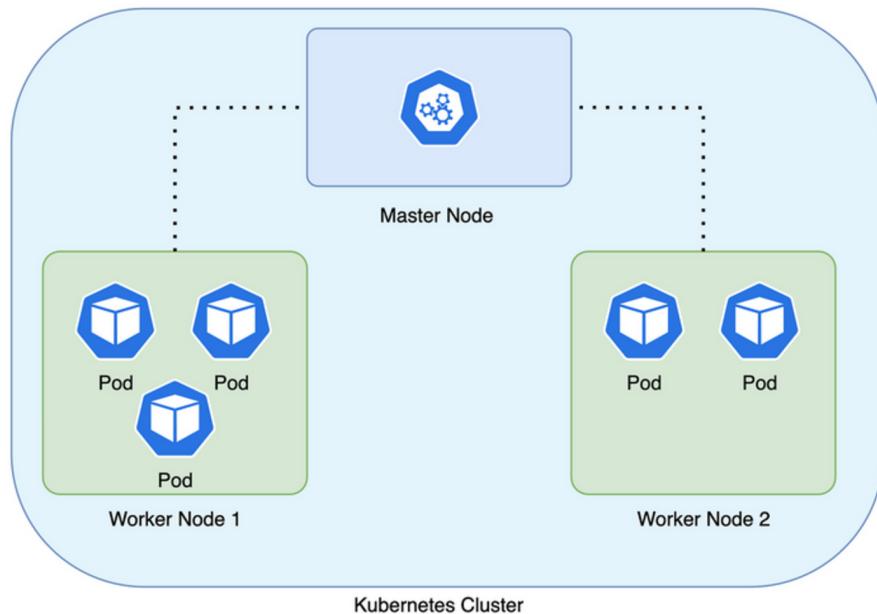
Un pod Kubernetes est un groupe de conteneurs qui fonctionnent ensemble, partageant le même réseau et le même stockage sur un nœud. Ils représentent l'unité de déploiement la plus petite et la plus simple dans Kubernetes.

Cluster

Un cluster Kubernetes est un groupe de nœuds (machines physiques ou virtuelles) qui exécutent des applications conteneurisées de manière efficace, automatisée, distribuée et évolutive. Il contient au minimum :

- Un nœud principal (master node) qui contrôle l'état du cluster et assigne les tâches
- Un ou plusieurs nœuds de travail (worker nodes) qui exécutent les applications

Des pods qui encapsulent les conteneurs de l'application.



Deploiement

Un déploiement Kubernetes est un objet de niveau supérieur qui fournit des mises à jour déclaratives pour les pods et les réplicas. Il gère le cycle de vie des déploiements, assurant la disponibilité de vos applications et la possibilité de mettre à jour les images de conteneur sans interruption. Les déploiements permettent de créer de nouveaux pods, de les mettre à l'échelle et de les mettre à jour en toute sécurité.

Service

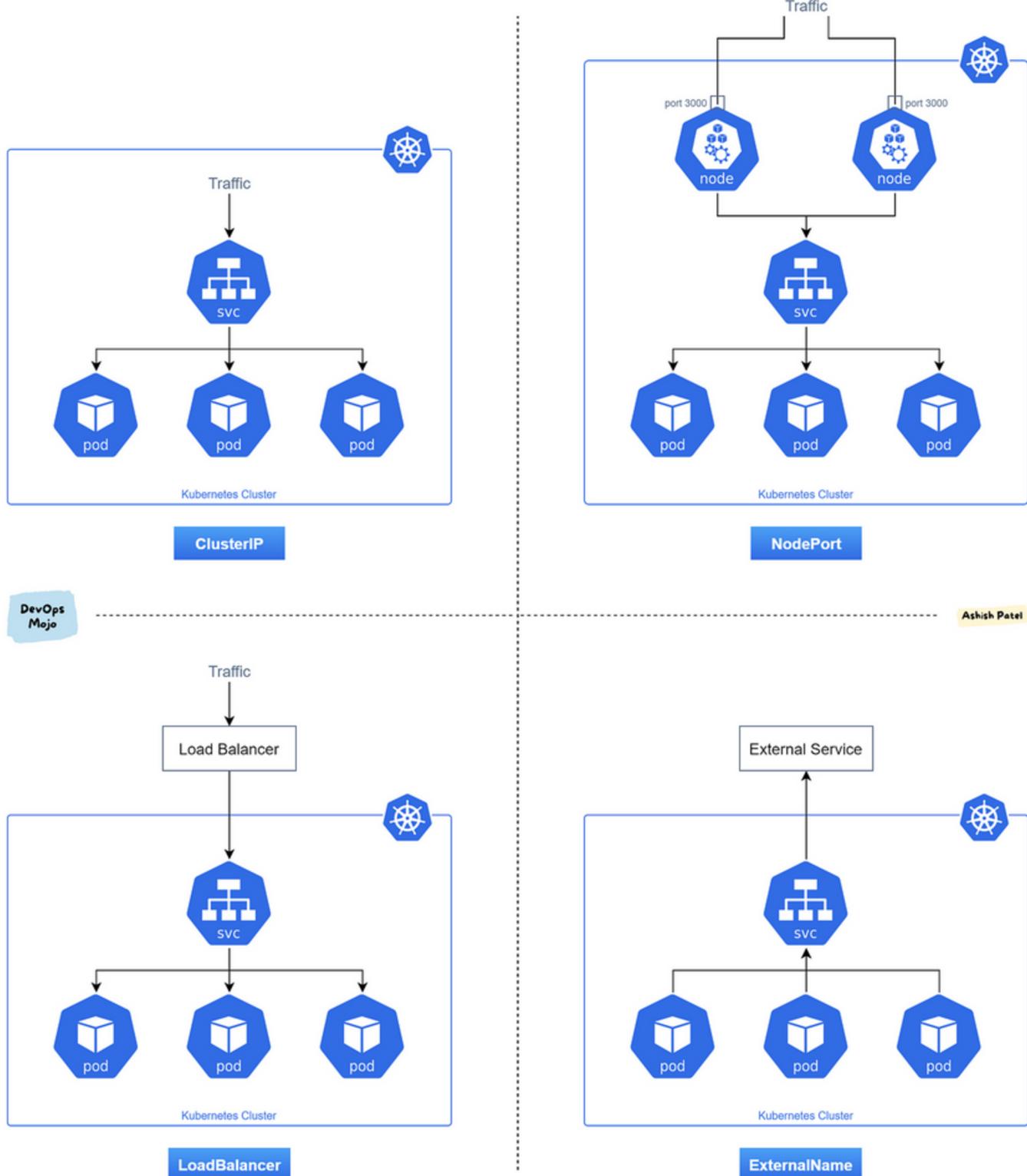
Un service Kubernetes est une abstraction qui définit un ensemble logique de pods et une politique pour y accéder. Les services permettent d'exposer des pods au réseau, facilitant ainsi la communication entre les différents composants d'une application. Chaque service se voit attribuer une adresse IP unique et un nom de domaine qui reste le même même si les pods sous-jacents changent.

• Types de services Kubernetes

ClusterIP : adresse IP virtuelle interne pour la communication entre pods

NodePort : expose un service sur un port statique de chaque nœud

LoadBalancer : utilise un équilibré de charge externe pour distribuer le trafic



• Les Objets Kubernetes

Un StatefulSet est un objet Kubernetes utilisé pour gérer des applications nécessitant un état persistant. Il permet de déployer des pods avec des identités stables et des volumes persistants, ce qui est essentiel pour des applications comme les bases de données. Les StatefulSets ne sont pas des pods eux-mêmes, mais ils gèrent la création et la gestion de plusieurs pods.

Un ConfigMap est un objet API Kubernetes utilisé pour stocker des données de configuration non sensibles sous forme de paires clé-valeur. Les données d'un ConfigMap peuvent être consommées par les pods sous forme de variables d'environnement,

Un secret est un objet API Kubernetes utilisé pour stocker des données sensibles, comme des mots de passe, des jetons OAuth ou d'autres données confidentielles. Contrairement aux ConfigMaps qui stockent des données non sensibles, les secrets sont conçus pour fournir une certaine protection des données.

Un volume dans Kubernetes est un objet qui représente un répertoire accessible par les conteneurs d'un pod. Contrairement aux systèmes de fichiers des conteneurs, qui sont éphémères et perdent les données lorsque le conteneur est redémarré ou arrêté, les volumes permettent de conserver les données au-delà de la durée de vie d'un conteneur.

3.9 Minikube



• Qu'est-ce que Minikube

Maintenant passant au minikube , pour cela j'ai fait l'introduction du kubernetes pour préparer la partie Minikube qui présente un outil puissant conçu pour permettre aux développeurs de créer un cluster Kubernetes local. Il facilite l'apprentissage et le développement d'applications Kubernetes sans nécessiter de ressources étendues. Minikube simplifie le processus d'exécution de Kubernetes en fournissant une interface en ligne de commande simple pour démarrer, gérer et supprimer des clusters.

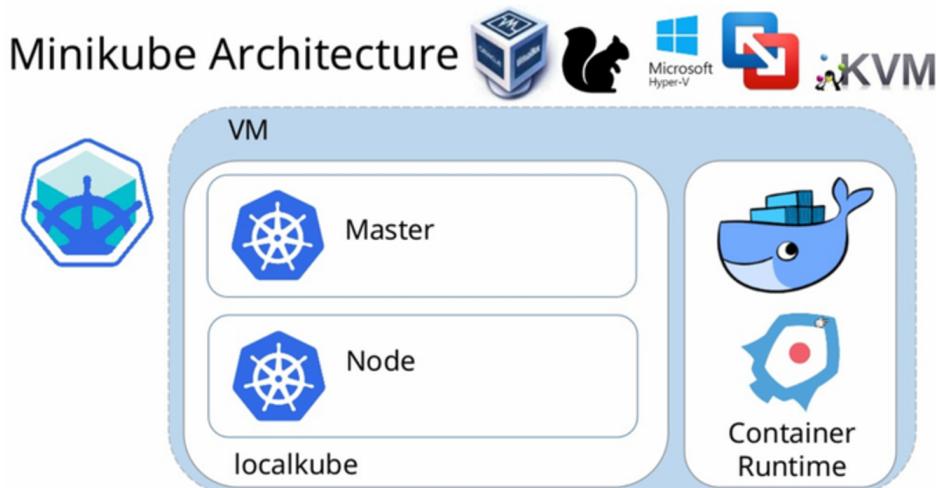
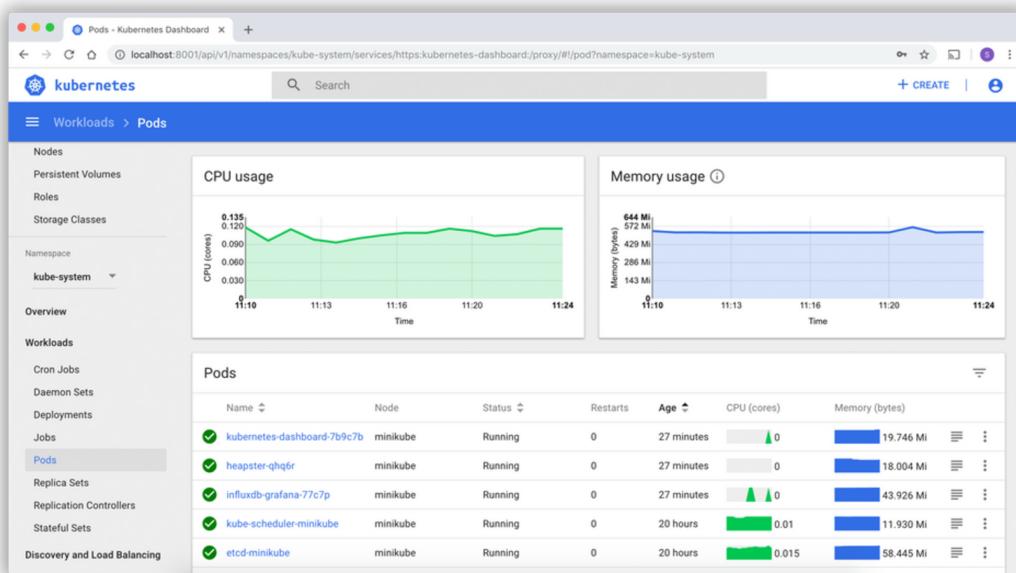
• Caractéristiques Clés du Minikube concernant le Testing

Cluster à Nœud Unique : Minikube crée un cluster Kubernetes à un seul nœud, où les processus maître et ouvrier s'exécutent sur la même machine.

Support Multiplateforme : Il peut être installé sur différents systèmes d'exploitation, notamment Windows, macOS et Linux.

Extensions : Minikube prend en charge diverses extensions pour améliorer ses fonctionnalités, comme l'activation du tableau de bord Kubernetes ou du serveur de métriques.

Efficacité des Ressources : Il est conçu pour fonctionner sur des machines avec des ressources limitées, ce qui le rend accessible aux développeurs sans matériel haute performance.



- **Limitations de Minikube concernant le déploiement**

Absence de Support IPv6 : Minikube ne prend pas en charge IPv6, ce qui peut être un inconvénient pour certains projets nécessitant cette fonctionnalité

Non destiné à la Production : Minikube est principalement conçu pour le développement et les tests. Il n'est pas adapté pour des déploiements en production, car il ne fournit pas les fonctionnalités avancées nécessaires pour gérer des applications à grande échelle

Cluster à Nœud Unique : Minikube ne permet de créer qu'un cluster à un seul nœud. Cela signifie qu'il ne peut pas simuler des environnements de production qui nécessitent plusieurs nœuds pour la haute disponibilité et la répartition de charge

Mes notes pour l'approche que j'ai adapté :

Dans ce projet, il n'était pas nécessaire d'adopter l'approche Kubernetes, car mon objectif était de tester mes microservices, donc Minikube était l'outil idéal pour cela. D'autre part, c'était mon premier pas dans le domaine du DevOps et du cloud, et c'était aussi ma première fois en tant que développeur à base de microservices et de Spring Cloud. Minikube s'est révélé très adaptable à mon processus pour lancer mes services sur les pods et m'a permis de comprendre la notion et le mécanisme de Kubernetes et de Minikube.

CHAPITRE 4: AUTOMATISATION



4.1 Jenkins

- **Qu'est-ce que Jenkins**

Jenkins est un serveur d'intégration continue (CI) open source qui gère et contrôle plusieurs étapes du processus de livraison de logiciels, y compris la construction, la documentation, les tests automatisés, l'emballage et l'analyse statique du code. C'est un outil DevOps très populaire utilisé par des milliers d'équipes de développement.

Jenkins est souvent déclenché par des modifications de code dans des dépôts comme GitHub, Bitbucket et GitLab, et s'intègre avec des outils de construction comme Maven et Gradle. Il prend également en charge l'utilisation de technologies de conteneur comme Docker et Kubernetes pour les tests et l'emballage des versions logicielles, mais ce n'est ni une solution native Kubernetes ni une solution CI native aux conteneurs.

- **Caractéristiques clés de Jenkins**

Réduction des efforts de codage répété : Les travaux Jenkins peuvent encapsuler le code de la ligne de commande dans des clics de bouton GUI, économisant ainsi des centaines de lignes de code.

Intégration de travaux individuels : Les travaux Jenkins peuvent être combinés à l'aide du plugin pipeline, permettant des combinaisons à la fois séquentielles et parallèles.

Intégration avec Slack : Jenkins peut être intégré à Slack pour partager des informations sur les activités déclenchées, leur temps, le nom des utilisateurs, les résultats, etc. avec l'équipe.

Automatisation : Jenkins peut être utilisé pour automatiser des tâches répétitives telles que la sauvegarde/restauration de bases de données, l'allumage/extinction de machines, la collecte de statistiques sur des services, etc.

• Limitations de Jenkins

Complexité de Configuration:

La configuration de Jenkins peut être complexe, surtout dans des environnements cloud. Les intégrations avec des outils modernes nécessitent souvent des personnalisations et un entretien des plugins, ce qui peut entraîner des erreurs et un surcroît de travail pour les équipes DevOps.

Gestion des Plugins:

Jenkins dispose d'un vaste écosystème de plugins, mais cela peut également poser des problèmes. La gestion de près de 2000 plugins peut devenir écrasante, et beaucoup d'entre eux sont abandonnés ou non maintenus, ce qui peut entraîner des incompatibilités et des problèmes de sécurité.

Sécurité:

Jenkins a été critiqué pour ses vulnérabilités de sécurité. Les mises à jour régulières des plugins et de Jenkins lui-même sont nécessaires pour éviter les failles de sécurité, mais cela peut être difficile à gérer, surtout si les plugins ne sont pas compatibles avec les nouvelles versions de Jenkins.

Exigences en matière de Compétences (Script Groovy)

Jenkins utilise Groovy pour ses pipelines, une langue qui n'est pas largement adoptée. Cela peut rendre le codage et la maintenance des scripts plus difficiles, surtout pour les équipes qui ne sont pas familières avec ce langage.

Mes notes pour l'approche que j'ai adapté :

J'ai utilisé Jenkins basé sur un conteneur, donc la configuration de Jenkins sera différente de celle en local.

Pourquoi cette approche ?: Tout simplement pour bien manipuler les conteneurs, car j'ai constaté que la flexibilité offerte par Docker est idéale pour inspecter et déboguer les conteneurs. De plus, c'est mon premier projet où j'utilise Docker de cette manière, donc pour me familiariser avec le concept de conteneurs, j'ai opté pour Jenkins basé sur un conteneur.

D'autre part, cette approche présente des avantages tels que l'isolation et la sécurité, ainsi que l'intégration avec l'écosystème DevOps.

CHAPITRE 5: Analyse des services



5.1 Sonarqube

- **Qu'est-ce que Sonarqube**

SonarQube est un outil d'analyse statique de code open-source développé par SonarSource, utilisé pour mesurer et améliorer la qualité du code dans divers projets logiciels.

- **Fonctionnalités de SonarQube**

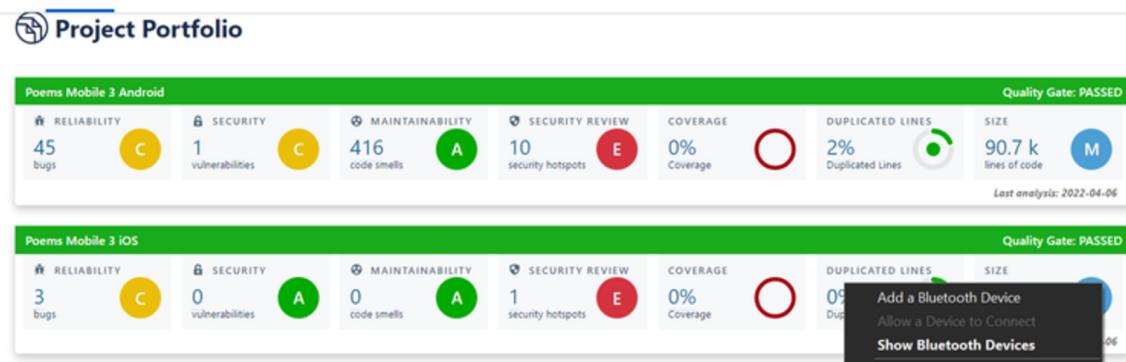
Analyse de la qualité du code : SonarQube évalue le code en fonction de plusieurs critères, notamment :

- Respect des règles de codage
- Documentation et commentaires
- Couverture des tests unitaires
- Détection de duplications de code
- Identification de vulnérabilités potentielles, classées par degré d'importance (mineure, majeure, bloquante)

Rapports et métriques : Il génère des rapports détaillés qui aident les développeurs à identifier les points à corriger, comme le code mort, les bugs potentiels, et les violations de standards de codage.

Compatibilité avec divers langages : SonarQube prend en charge plus de 25 langages de programmation, y compris Java, C#, PHP, et JavaScript, et s'adapte aux versions de ces langages pour signaler le code obsolète.

Intégration continue : Il peut être intégré dans des pipelines CI/CD, facilitant ainsi l'analyse régulière du code tout au long du cycle de développement.



• Fonctionnement

SonarQube se compose de deux éléments principaux :

- **Un moteur d'analyse** (le scanner) : Ce composant est installé localement sur la machine du développeur et est responsable de l'analyse du code.
- **Un serveur centralisé** : Il stocke les résultats des analyses et génère des rapports accessibles via une interface web

• Limitations de SonarQube

Complexité de configuration initiale

Bien que SonarQube soit relativement facile à installer avec Docker, la configuration initiale peut être complexe, surtout pour intégrer l'analyse dans un pipeline CI/CD existant. Cela nécessite de bien comprendre les différentes options de configuration disponibles

Coût élevé pour les grandes équipes

Bien que l'édition Community soit gratuite, les éditions payantes deviennent rapidement coûteuses pour les très grandes équipes avec beaucoup de lignes de code. Le coût dépend du nombre de lignes de code et des fonctionnalités avancées requises.

Mes notes pour l'approche que j'ai adapté :

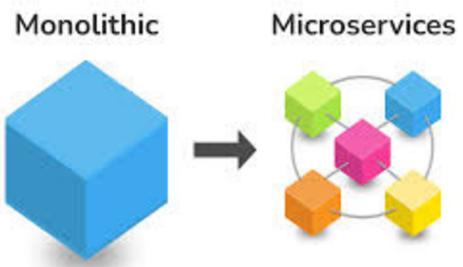
J'ai utilisé **SonarQube** basé sur un conteneur, lorsque vous lancez SonarQube, le serveur utilise une base de données H2 qui stocke les données localement et les perd lorsque le serveur est arrêté.

J'ai préféré l'approche SonarQube basée sur un **conteneur** plutôt que de le télécharger et de l'exécuter en local, pour les mêmes raisons que pour Jenkins basé sur un conteneur.

CHAPITRE 6: L'architecture Microservices

- **Concept des microservices**

Les microservices sont une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments plus simples et indépendants.



- **Caractéristiques clés des microservices**

Décomposition en éléments simples : Les microservices décomposent une application en plusieurs services indépendants, chacun ayant une responsabilité spécifique. Cela permet une meilleure gestion, une évolutivité plus facile et une réduction des risques de défaillance globale.

Indépendance : Chaque microservice peut être développé, testé et déployé séparément, ce qui permet l'utilisation de langages de programmation et de frameworks différents pour chaque service.

Évolutivité : Les microservices peuvent être mis à l'échelle de manière indépendante, ce qui signifie que seul le service qui nécessite plus de ressources est augmenté, sans affecter les autres services.

- **Architecture et Fonctionnement**

Service d'enregistrement : Dans une architecture microservices, un service d'enregistrement est souvent utilisé pour permettre aux microservices de s'enregistrer et de publier leurs informations de localisation (adresse IP et port). Cela facilite la découverte et la communication entre les services.

Configuration centralisée : Pour éviter de créer des fichiers de configuration pour chaque microservice, une configuration centralisée est souvent mise en place. Les microservices se connectent à un service de configuration pour récupérer leurs paramètres de configuration, ce qui permet des mises à jour en temps réel sans redémarrage.

Communication entre services : Les microservices communiquent généralement via des API existantes comme REST, SOAP, AMQP, etc. Il est recommandé d'utiliser un bus de messages pour éviter les interactions directes entre les services et ainsi maintenir leur indépendance.

- **Bonnes Pratiques et Outils**

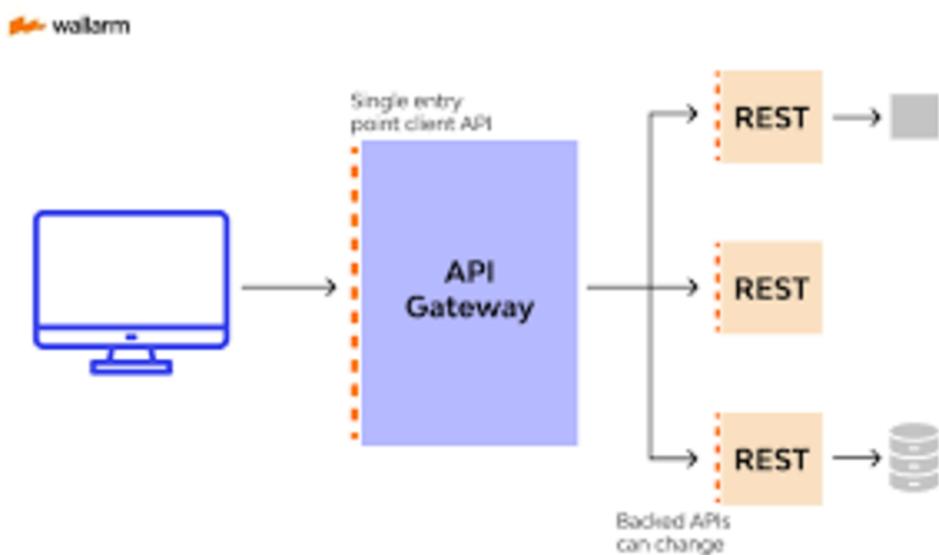
Infrastructure : Une grande infrastructure est nécessaire pour supporter les microservices. Cela inclut des serveurs hôtes, des bases de données et des systèmes d'exploitation. Des outils de gestion de configuration comme Ansible, Chef ou Puppet sont utilisés pour configurer et gérer ces infrastructures.

Conteneurisation : Les conteneurs, comme ceux créés avec Docker, sont souvent utilisés pour déployer les microservices. Cela permet une isolation et une portabilité accrues des applications.

Déploiement et gestion : Des outils comme Mesos et Marathon sont utilisés pour gérer et déployer les microservices sur un cluster de serveurs. Ces outils permettent de démarrer et de relancer automatiquement les instances de services en cas de défaillance.

- **Principe de Découplage**

Découplage des services : Les microservices doivent être conçus pour fonctionner de manière indépendante. Ils ne doivent pas communiquer directement entre eux, mais plutôt via un service d'intégration comme un bus de messages. Cela permet une évolution indépendante des services et une meilleure tolérance aux pannes.



- **Limitations des microservices**



Déploiement et Versioning

Automatisation des Déploiements : Les microservices nécessitent des déploiements automatisés pour gérer efficacement le nombre de services. Cela demande une intégration de technologies d'automatisation comme GitHub, Jenkins et Terraform.

Challenges de Versioning : La coordination des déploiements et la gestion des versions entre les différents services peuvent être complexes, entraînant des problèmes de compatibilité.



Intégration et Dépendances

Surcharge d'Intégration : Les microservices doivent communiquer entre eux via des API, ce qui peut entraîner des surcharges d'intégration et des problèmes de dépendances.

Problèmes de Communication : La communication entre les services via le réseau peut augmenter la latence, les surcharges réseau et les points de défaillance potentiels.

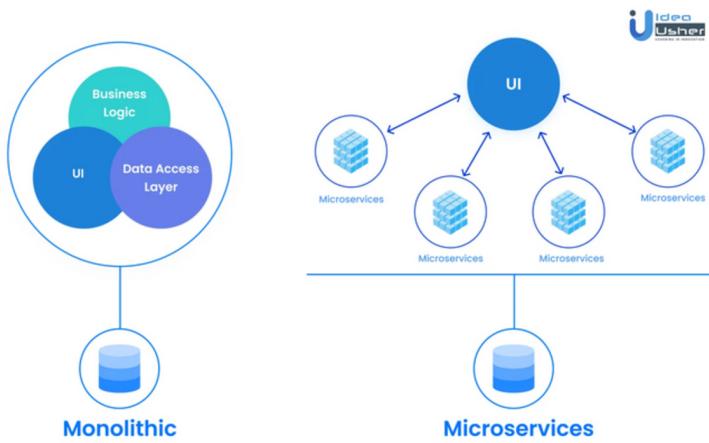


Performance et Ressources

Performance Diminuée : Les microservices consomment plus de ressources (mémoire, cycles d'horloge et bande passante réseau) que les architectures monolithiques. Cela est dû aux appels réseau et à la sérialisation/désérialisation des données.

Coûts Accrus : L'utilisation de plusieurs microservices nécessite plus de ressources, ce qui peut augmenter les coûts, notamment dans les environnements cloud où les ressources sont facturées en fonction de leur utilisation.

• Différence entre l'architecture microservice et monolithique



Structure de l'Application:

Monolithique : Une application monolithique est construite comme une unité unique et indivisible. Tous les composants logiciels, y compris l'interface utilisateur, l'application côté serveur et la base de données, sont regroupés dans une seule base de code.

Microservices : Les architectures de microservices décomposent l'application en petits services indépendants, chacun jouant un rôle unique et communiquant via des API bien définies.

Développement et Déploiement:

Monolithique : Les applications monolithiques sont plus simples à développer et à déployer en raison de leur nature singulière.

Microservices : Les microservices nécessitent plus de planification et de conception initiale, mais permettent un développement et un déploiement indépendants des services, ce qui accélère les versions de fonctionnalités et réduit le risque d'échec.

Évolutivité:

Monolithique : Les applications monolithiques peuvent être mises à l'échelle, mais uniquement de manière horizontale, en exécutant plusieurs copies de l'ensemble de l'application. Cela peut être gourmand en ressources et inefficace.

Microservices : Les microservices permettent une mise à l'échelle indépendante des services en fonction de leurs besoins spécifiques, ce qui se traduit par une allocation efficace des ressources et une amélioration des performances.

 En résumé, les architectures monolithiques sont appropriées pour les applications simples et légères, tandis que les architectures de microservices conviennent mieux aux applications complexes et évolutives qui nécessitent une grande flexibilité et une mise à l'échelle efficace.

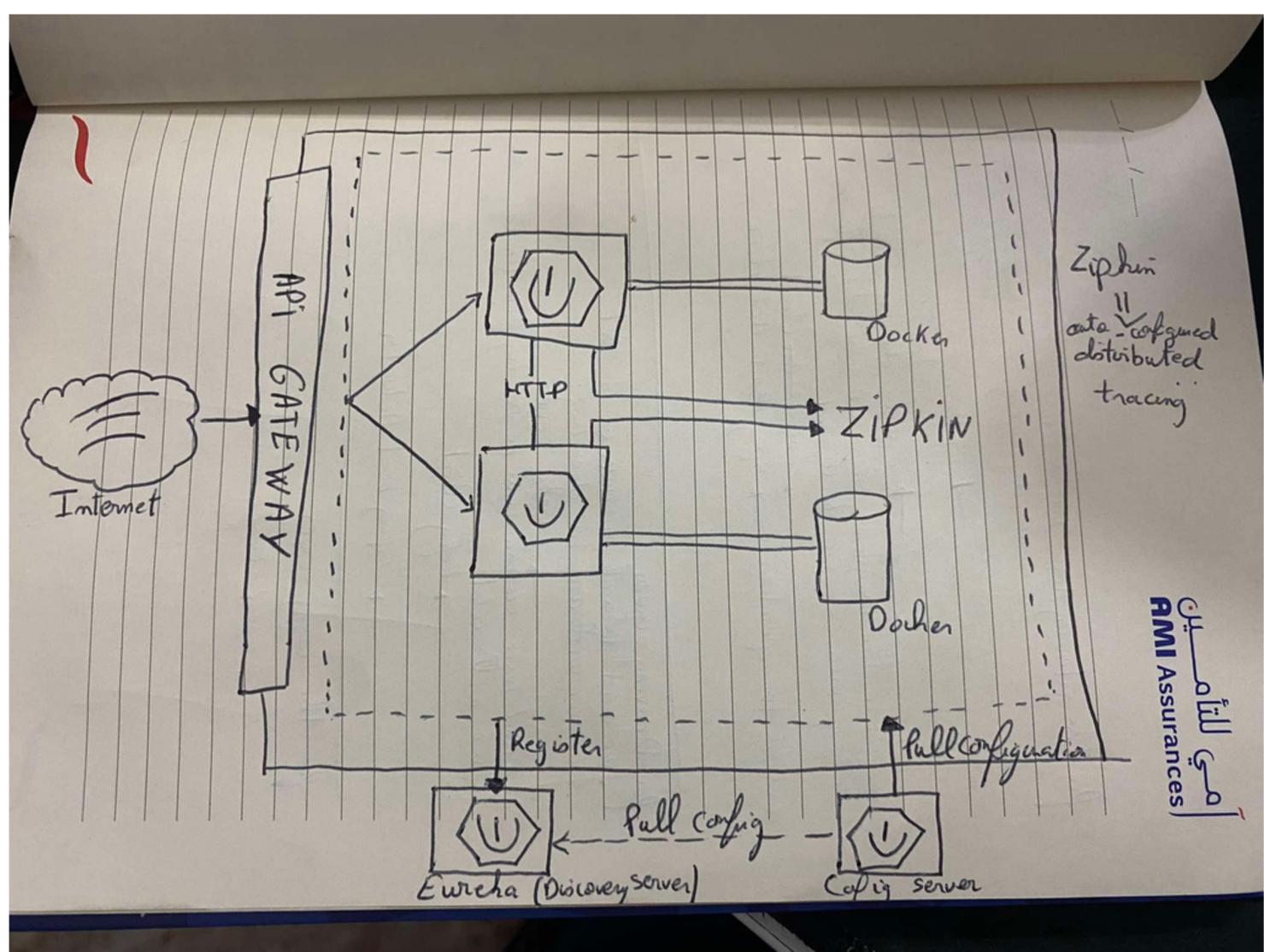
CHAPITRE 7: Conception et Réalisation

• Partie Développement avec Spring Cloud et Microservices

Tout d'abord, je vais présenter le projet de microservices que j'ai développé. Cela représentait la première étape de mon parcours vers la création d'un pipeline automatisé.

Pour ce faire, j'ai travaillé avec Spring Cloud basé sur une architecture de microservices. J'ai développé le Config Server, le Discovery Server et l'API Gateway, qui sont les éléments principaux d'une architecture microservices, suivis des autres services tels que Assurance, AssurancePolicy. De plus, pour optimiser l'utilisation de l'architecture microservices, j'ai intégré deux types de bases de données : PostgreSQL et H2, ainsi qu'un serveur Zipkin pour le traçage, comme le montre la figure et pour plus de ressources, voici le lien vers mon répertoire sur GitHub :

<https://github.com/benammarfares/Assurance-MicroService>



- **Partie Dockerisation des microservices :**

Dans cette partie, j'ai créé des images en me basant sur les services. Pour cela, j'ai installé Docker Desktop afin d'utiliser le Docker Daemon et ainsi générer des images à partir du Dockerfile présent dans chaque service.

```
FROM openjdk:17
COPY /target/*.jar assurance.jar
EXPOSE 8070
ENTRYPOINT ["java", "-jar", "assurance.jar"]
```

- FROM openjdk:17 : Cette ligne spécifie l'image de base à utiliser pour construire l'image Docker. Ici, j'ai choisi l'image openjdk:17, qui contient le Java Development Kit (JDK) version 17, correspondant à la version de Java que j'ai utilisée.
- COPY /target/*.jar assurance-service.jar : Cette ligne copie le fichier .jar (généralement produit par un processus de construction comme Maven ou Gradle, dans mon cas j'ai utilisé Maven) depuis le répertoire target de mon service vers le système de fichiers de l'image Docker, en le renommant assurance-service.jar.
- EXPOSE 8070 : Cette ligne indique que le conteneur écoute sur le port 8070. Cela permet à d'autres conteneurs ou à l'hôte d'accéder à ce port, facilitant ainsi la communication avec l'application.
- ENTRYPOINT ["java", "-jar", "assurance-service.jar"] : Cette ligne définit la commande qui sera exécutée lorsque le conteneur sera démarré. Ici, elle exécute la commande java -jar assurance-service.jar, ce qui lance mon application Java contenue dans le fichier JAR que j'ai copié précédemment.

• Partie Communication entre les conteneurs

Dans cette partie, je vais lancer les images que j'ai déjà créées, ce qui me permettra d'exécuter les services en conteneurs plutôt que dans IntelliJ. Pour cela, il est nécessaire de passer par Docker Compose afin d'assurer la communication et le lancement des conteneurs.

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "Devops-Microservice". The "assurance" module is selected, displaying its Dockerfile and target directory.
- Code Editor:** The "assurancePolicy.yml" file is open, showing the Docker Compose configuration for the "assurance" service. It includes definitions for PostgreSQL, Zipkin, and a Config Server.
- Docker Desktop:** On the right, the Docker Desktop interface lists the running containers:
 - gateway (devops-microservice) - Running, port 8222:8222
 - assurance-service (devops-microservice) - Running, port 8070:8070
 - assurance-Policy... (devops-microservice) - Running, port 8060:8060
 - discovery-server (devops-microservice) - Running, port 8761:8761
 - devops-microservi... (postgres:latest) - Running, port 60407:5432
 - zipkin (openzipkin/zipkin) - Running, port 9411:9411
 - config-server (devops-microservice) - Running, port 8888:8888
- Services Tab:** Shows the configuration for an "HTTP Request" named "GettingPolicyFromAssurance".
- Status Bar:** RAM 4.22 GB CPU 0.58%

Pour achever ce processus, j'ai créé les fichiers JAR de chaque service que j'ai intégrés via le fichier docker-compose.yml. J'ai également implémenté des variables d'environnement et un mécanisme de vérification de l'état (Health Check) pour que les services puissent se connecter au Discovery Server. L'élément le plus important est de s'assurer qu'ils prennent leur configuration depuis le Config Server.

System Status

Environment	test	Current time	2024-08-02T17:20:36 +0000
Data center	default	Uptime	00:03
		Lease expiration enabled	false
		Renews threshold	6

DS Replicas

[config-server](#)

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ASSURANCE	n/a (1)	(1)	UP (1) - 422c9b05173b.assurance:8070
ASSURANCEPOLICY	n/a (1)	(1)	UP (1) - 13803a26bff2.assurancePolicy:8060
GATEWAY	n/a (1)	(1)	UP (1) - be9161236a09.gateway:8222

General Info

Name	Value
total-avail-memory	93mb
num-of-cpus	12
current-memory-usage	46mb (49%)

Enregistrement des conteneurs dans le serveur Eureka

- L'approche est différente de celle utilisée dans un environnement de développement classique, comme celui que j'ai utilisé (IntelliJ). Il était donc nécessaire de comprendre des notions clés dans Docker Compose, telles que les dépendances, le Health Check, les volumes pour la base de données et le réseau.

The screenshot shows two windows side-by-side. On the left, the Docker Desktop interface displays the logs for the 'gateway' container. The logs show various INFO messages from the Netflix Discovery Client and Netty Web Server, indicating the application is running and discovering services. On the right, a browser window shows the Zipkin UI at localhost:9411. It visualizes a trace for a 'http post' request from the 'gateway' service. The trace spans from a 'Server Start' event at 0ms to a 'Server Finish' event at 160.422ms. Annotations for 'Server Start' and 'Server Finish' are shown with their respective start and end times and addresses (192.168.192.8). The Zipkin UI also includes a 'SPAN TABLE' and a 'Tags' section with entries like 'exception: none', 'http.url: /api/v1/assurance/save', 'method: POST', 'outcome: SUCCESS', 'status: 201', and 'uri: UNKNOWN'.

Inspection du serveur Zipkin pour le traçage d'une API





• Partie Orchestration des conteneurs

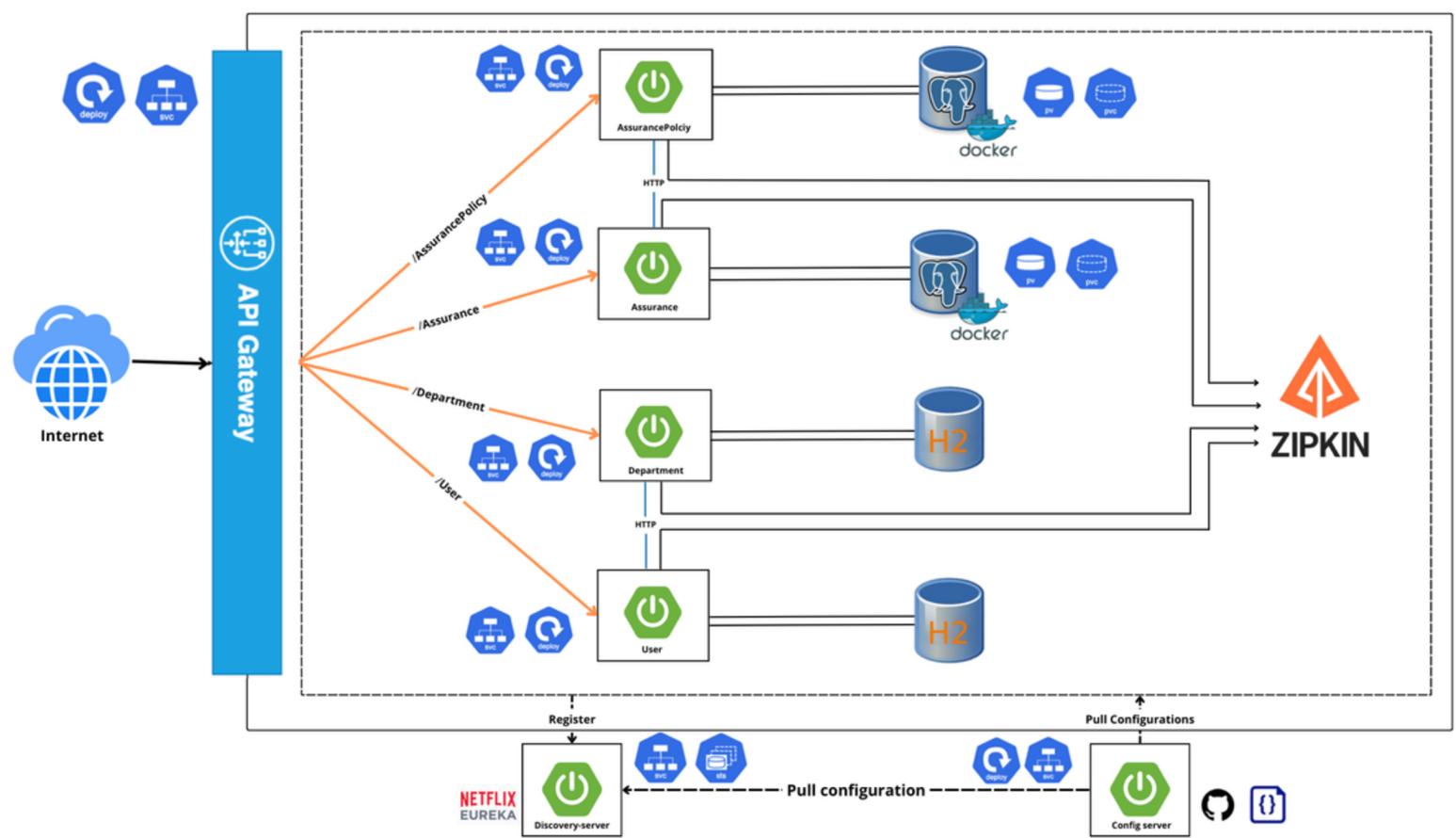
Dans cette partie, je vais utiliser **Minikube** pour l'orchestration de mes conteneurs.

Étant donné que j'ai bien maîtrisé les deux parties précédentes, j'ai ajouté deux nouveaux services à mon architecture (User et Department) afin d'explorer la flexibilité de l'architecture microservices. Ces deux nouveaux services utiliseront H2 comme base de données, tandis que les deux anciens services continueront d'utiliser PostgreSQL.

Ici le lien vers mon répertoire sur GitHub pour ce travail :

- <https://github.com/benammarfares/K8s-Microservice-SpringCloud>
- <https://github.com/benammarfares/Kubernetes-yml-for-service-deploy>

Conception pour cette partie



Préparation de ma machine virtuelle sur VMware Workstation :

- Dans cet exemple, j'ai utilisé Ubuntu 22.04 (version stable) pour le lancement, car Minikube dépend de la virtualisation.

```
fares@fares-virtual-machine:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.4 LTS
Release:        22.04
Codename:       jammy
fares@fares-virtual-machine:~$
```

Installation de Docker :

- J'ai installé Docker afin de pouvoir l'associer à Minikube par la suite comme driver.

```
fares@fares-virtual-machine:~$ docker version
Client: Docker Engine - Community
  Version: 27.1.2
  API version: 1.46
  Go version: go1.21.13
  Git commit: d01f264
  Built: Mon Aug 12 11:50:12 2024
  OS/Arch: linux/amd64
  Context: default

Server: Docker Engine - Community
  Engine:
    Version: 27.1.2
    API version: 1.46 (minimum version 1.24)
    Go version: go1.21.13
    Git commit: f9522e5
    Built: Mon Aug 12 11:50:12 2024
    OS/Arch: linux/amd64
    Experimental: false
  containerd:
    Version: 1.7.20
    GitCommit: 8fc6bcff51318944179630522a095cc9dbf9f353
  runc:
    Version: 1.1.13
    GitCommit: v1.1.13-0-g58aa920
  docker-init:
    Version: 0.19.0
    GitCommit: de40ad0
fares@fares-virtual-machine:~$
```

Installation de Minikube :

- Dans cette partie, j'ai installé Minikube sur ma machine virtuelle. Pour pouvoir l'utiliser, il était nécessaire de spécifier un driver lors du lancement de Minikube , dans mon cas, j'ai utilisé Docker.

```
fares@fares-virtual-machine:~$ minikube start --driver=docker
minikube v1.33.1 on Ubuntu 22.04
Using the docker driver based on existing profile
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.44 ...
Restarting existing docker container for "minikube" ...
This container is having trouble accessing https://registry.k8s.io
To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
Verifying Kubernetes components...
  ▪ Using image docker.io/kubernetesui/dashboard:v2.7.0
  ▪ Using image docker.io/kubernetesui/metrics-scraper:v1.0.8
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
  Some dashboard features require the metrics-server addon. To enable all features please run:

      minikube addons enable metrics-server

  Enabled addons: default-storageclass, storage-provisioner, dashboard
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

D'autre part, j'ai installé kubectl, bien que cela ne soit pas nécessaire pour le moment, car on peut utiliser un alias pour kubectl dans Minikube. En effet, kubectl est inclus en tant que dépendance dans Minikube.

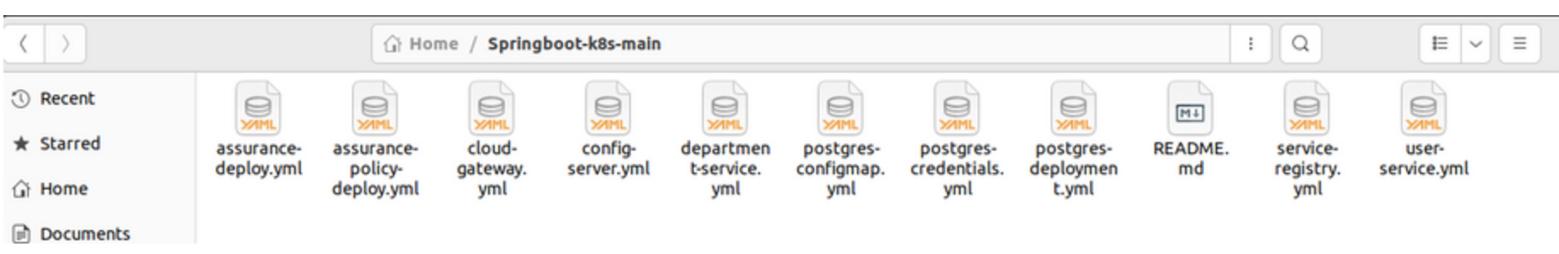
```
fares@fares-virtual-machine:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

```
fares@fares-virtual-machine:~$ kubectl version
Client Version: v1.31.0
Kustomize Version: v5.4.2
Server Version: v1.30.0
```

Tirer les images depuis Docker Hub dans la machine virtuelle

```
fares@fares-virtual-machine:~$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
fares121/user-service    1.0.0   96a2f5588a08  6 days ago   546MB
fares121/department-service  1.0.0   3b83ba055d01  6 days ago   546MB
fares121/assurancepolicy-service  1.0.0   a1ae5989b9e2  6 days ago   545MB
fares121/assurance-service    1.0.0   b718f670c657  6 days ago   548MB
fares121/cloud-gateway     1.0.0   bf59c30d8e32  6 days ago   522MB
fares121/service-registry   1.0.0   18efb130efa6  6 days ago   527MB
fares121/cloud-config-server 1.0.0   87590a09d36c  6 days ago   524MB
```

Créer un répertoire pour implémenter les fichiers Kubernetes (service, déploiement, secret, PV)



Accéder au répertoire et exécuter tous les fichiers

```
fares@fares-virtual-machine:~/Springboot-k8s-main$ kubectl apply -f .
service/assurance unchanged
deployment.apps/assurance unchanged
service/assurance-policy unchanged
deployment.apps/assurance-policy unchanged
deployment.apps/cloud-gateway-app unchanged
service/cloud-gateway-svc unchanged
deployment.apps/cloud-config-server-app unchanged
service/cloud-config-server-svc unchanged
deployment.apps/department-service-app unchanged
service/department-service-svc unchanged
configmap/postgres-conf unchanged
secret/postgres-credentials unchanged
service/postgres unchanged
persistentvolumeclaim/postgres-pv-claim unchanged
deployment.apps/postgres unchanged
configmap/eureka-cm unchanged
service/eureka unchanged
statefulset.apps/eureka unchanged
service/eureka-lb unchanged
deployment.apps/user-service-app unchanged
service/user-service-svc unchanged
```

Récupère la liste de toutes les ressources Kubernetes

```

fares@fares-virtual-machine:~$ cd Springboot-k8s-main
fares@fares-virtual-machine:~/Springboot-k8s-main$ alias kubectl="minikube kubectl --"
fares@fares-virtual-machine:~/Springboot-k8s-main$ kubectl get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/assurance-855c6cc489-fs2rj  1/1    Running   2 (38s ago)  17h
pod/assurance-policy-84748ddf9c-xbk9j  1/1    Running   2 (38s ago)  17h
pod/cloud-config-server-app-f95db9898-m7lcb  1/1    Running   2 (38s ago)  17h
pod/cloud-gateway-app-5fb64b6446-d22hw  1/1    Running   2 (38s ago)  17h
pod/department-service-app-5f898bd5fb-sxwsn  1/1    Running   2 (38s ago)  17h
pod/eureka-0                    1/1    Running   2 (38s ago)  17h
pod/postgres-78d885c78d-6rw9d  1/1    Running   2 (38s ago)  17h
pod/user-service-app-57dbc6fcf6-bsf4j  1/1    Running   2 (38s ago)  17h

NAME              TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/assurance   ClusterIP   <none>          80/TCP          17h
service/assurance-policy   ClusterIP   <none>          80/TCP          17h
service/cloud-config-server-svc   ClusterIP   <none>          80/TCP          17h
service/cloud-gateway-svc     LoadBalancer  <pending>       80:30950/TCP  17h
service/department-service-svc   ClusterIP   <none>          80/TCP          17h
service/eureka       ClusterIP   None            <none>          8761/TCP      17h
service/eureka-lb     NodePort    <none>          80:31466/TCP  17h
service/kubernetes   ClusterIP   <none>          443/TCP         9d
service/postgres     ClusterIP   None            <none>          5432/TCP      17h
service/user-service-svc   ClusterIP   <none>          80/TCP          17h

NAME              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/assurance  1/1      1           1           17h
deployment.apps/assurance-policy  1/1      1           1           17h
deployment.apps/cloud-config-server-app  1/1      1           1           17h
deployment.apps/cloud-gateway-app  1/1      1           1           17h
deployment.apps/department-service-app  1/1      1           1           17h
deployment.apps/postgres   1/1      1           1           17h
deployment.apps/user-service-app  1/1      1           1           17h

NAME              DESIRED  CURRENT  READY   AGE
replicaset.apps/assurance-855c6cc489  1        1        1        17h
replicaset.apps/assurance-policy-84748ddf9c  1        1        1        17h
replicaset.apps/cloud-config-server-app-f95db9898  1        1        1        17h
replicaset.apps/cloud-gateway-app-5fb64b6446  1        1        1        17h
replicaset.apps/department-service-app-5f898bd5fb  1        1        1        17h
replicaset.apps/postgres-78d885c78d  1        1        1        17h
replicaset.apps/user-service-app-57dbc6fcf6  1        1        1        17h

NAME              READY   AGE
statefulset.apps/eureka  1/1    17h

```

Exposer le service Eureka pour visualiser les services enregistrés :

```

fares@fares-virtual-machine:~/Springboot-k8s-main$ minikube service eureka-lb
|-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|-----|-----|-----|
| default | eureka-lb | 80 | http://[REDACTED]:31744 |
|-----|-----|-----|-----|
[!] Opening service default/eureka-lb in default browser...

```

Inspecter le serveur Eureka



HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2024-09-02T12:57:57 +0000
Data center	default	Uptime	00:57
		Lease expiration enabled	true
		Renews threshold	11
		Renews (last min)	12

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - cloud-gateway-app-857f57554d-9lpxf:API-GATEWAY:9191
ASSURANCE	n/a (1)	(1)	UP (1) - assurance-76b69b56cf-z5zzf:assurance:8070
ASSURANCEPOLICY	n/a (1)	(1)	UP (1) - assurance-policy-55b7bd9475-prjfl:assurancePolicy:8060
CONFIG-SERVER	n/a (1)	(1)	UP (1) - cloud-config-server-app-7f7855d956-85lvh:CONFIG-SERVER:9296
DEPARTMENT-SERVICE	n/a (1)	(1)	UP (1) - department-service-app-8c64bf8db-ng27x:DEPARTMENT-SERVICE:9001
USER-SERVICE	n/a (1)	(1)	UP (1) - user-service-app-579b97b874-8frzm:USER-SERVICE:9002

General Info

Exposer le service API Gateway en tant que Load Balancer

```
fares@fares-virtual-machine:~/Springboot-k8s-main$ minikube service cloud-gateway-svc
|-----|-----|-----|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----|-----|-----|-----|
| default | cloud-gateway-svc | 80 | http:// :30950 |
|-----|-----|-----|-----|
💡 Opening service default/cloud-gateway-svc in default browser...
```

Tous les APIs passeront par le Load Balancer. Pour cela, les autres services sont de type ClusterIP, car ils seront internes et non exposés. Le Load Balancer (API Gateway) jouera le rôle de répartiteur de charge et redirigera chaque API vers le service et le port destinés. Ainsi, cette API Gateway sera exposée.

Tester les APIs via Postman en utilisant l'adresse IP de la passerelle et le port NodePort

add assurance - My Workspace

File Edit View Help

← → Home Workspaces API Network Search Postman

New Import POST add • GET Show • POST add • GET Overv • GET Show • GET find u • GET Get / • GET http:// • User + No environment

My Workspace Collections + Assurance Collection / add assurance

POST add assurance

POST add assurance • GET Show all assurances • GET Get Assurance Policies By Ass... • POST add assurance-policy

POST add assurance-policy • GET Show all policies • CardioActivities • GET findAll • GET findById • POST create • PUT update • DEL delete • Department • POST add • GET find department by id • User • POST add user with department id

POST http://[REDACTED]:30950/api/v1/assurance/save

Params Authorization Headers (8) Body Scripts Settings

Body (Pretty, Raw, Preview, Visualize, Text)

1
 2 "claimName": "Accident Claim 1",
 3 "claimDate": "2023-04-01",
 4 "claimDescription": "Car accident on highway 1",
 5 "claimAmount": 5000.00,
 6 "claimStatus": "Submitted",
 7 "claimProcessedDate": "2023-04-10",
 8 "claimPayoutAmount": 3000.00
 9 }

Status: 201 Created Time: 3.48 s Size: 80 B Save as example

HTTP Assurance Collection / Get Assurance Policies By Assurance

GET http://[REDACTED]:30950/api/v1/assurance/getAssurance/1

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON

1
 2 "claimName": "Accident Claim 1",
 3 "claimDate": "2023-04-01T00:00:00.000+00:00",
 4 "claimDescription": "Car accident on highway 1",
 5 "claimAmount": 5000.0,
 6 "claimStatus": "Submitted",
 7 "claimProcessedDate": "2023-04-10T00:00:00.000+00:00",
 8 "claimPayoutAmount": 3000.0,
 9 "policyAssurance": [
 10 {
 11 "policyHolder": "Fares Ben Ammar",
 12 "policyType": "Health",
 13 "policyDetails": "Health insurance policy",
 14 "coverageDetails": "Full coverage",
 15 "amountOfAssurance": 100000.0,
 16 "policyStartDate": "2024-09-02T12:01:27.039+00:00",
 17 "policyEndDate": "2025-09-02T12:01:27.039+00:00",
 18 "policyStatus": "Active"
 19 }
 20]
 21 }

Postbot Runner Capture requests Cookies Vault Trash



امي للتأمين
AMI Assurances

Accéder au pod PostgreSQL et inspecter les tables à l'intérieur de la base de données :

NAME	READY	STATUS	RESTARTS	AGE
assurance-76b69b56cf-b9lxn	1/1	Running	0	7s
assurance-policy-55b7bd9475-5bl25	1/1	Running	0	7s
cloud-config-server-app-7f7855d956-grxbw	1/1	Running	0	7s
cloud-gateway-app-857f57554d-qwqt2	1/1	Running	0	7s
department-service-app-8c64bf8fdb-mxghc	1/1	Running	0	7s
eureka-0	1/1	Running	0	7s
postgres-78d885c78d-s2cz8	1/1	Running	0	7s
user-service-app-579b97b874-2dp5v	1/1	Running	0	6s

Se connecter à la base de données PostgreSQL en cours d'exécution sur un pod :

```
fares@fares-virtual-machine:~/Springboot-k8s-main$ kubectl exec -it postgres-78d885c78d-s2cz8 -- psql -h localhost -U postgres --password -p 5432 postgres
Password:
psql (16.4 (Debian 16.4-1.pgdg120+1))
Type "help" for help.

postgres=# \l
                                         List of databases
   Name    | Owner | Encoding | Locale Provider | Collate      | Ctype       | ICU Locale | ICU Rules | Access privileges
-----+-----+-----+-----+-----+-----+-----+-----+-----+
assurededb | postgres | UTF8 | libc | en_US.utf8 | en_US.utf8 | en_US.utf8 | en_US.utf8 | =c/postgres
postgres   | postgres | UTF8 | libc | en_US.utf8 | en_US.utf8 | en_US.utf8 | en_US.utf8 | postgres=CTc/postgres
template0  | postgres | UTF8 | libc | en_US.utf8 | en_US.utf8 | en_US.utf8 | en_US.utf8 | =c/postgres
template1  | postgres | UTF8 | libc | en_US.utf8 | en_US.utf8 | en_US.utf8 | en_US.utf8 | =c/postgres
(4 rows)
```

Accéder à la base de données assurededb et afficher les tables :

```
postgres=# \c assurededb
Password:
You are now connected to database "assurededb" as user "postgres".
assurededb=# \dt
          List of relations
 Schema |        Name        | Type | Owner
-----+-----+-----+-----+
 public | assurance_claim | table | postgres
 public | assurance_policy | table | postgres
(2 rows)
```

Afficher la table assurance_claim après l'ajout :

```
assurededb=# SELECT * FROM assurance_claim;
 claim_amount | claim_payout_amount | id | claim_date | claim_processed_date | claim_description | claim_name | claim_status
-----+-----+-----+-----+-----+-----+-----+-----+
 5000 | 3000 | 1 | 2023-04-01 00:00:00 | 2023-04-10 00:00:00 | Car accident on highway 1 | Accident Claim 1 | Submitted
(1 row)
```

Afficher la table assurance_policy après l'ajout :

```
assurededb=# SELECT policy_holder, policy_type, amount_of_assurance, amount_of_assurance, policy_end_date, policy_status, assurance_id FROM assurance_policy
 policy_holder | policy_type | amount_of_assurance | amount_of_assurance | policy_end_date | policy_status | assurance_id
-----+-----+-----+-----+-----+-----+-----+
 Fares Ben Ammar | Health | 100000 | 100000 | 2025-09-02 13:41:47.003 | Active | 1
 Jane Smith | Life | 200000 | 200000 | 2025-09-02 13:41:47.003 | Active | 2
 Fares | Comprehensive | 75000 | 75000 | 2024-12-31 00:00:00 | Active | 1
(3 rows)
```

Inspection du tableau de bord Minikube pour visualiser l'état des pods :

Workloads

Cron Jobs

Daemon Sets

Deployments

Jobs

Pods

Replica Sets

Replication Controllers

Stateful Sets

Deployments

Running: 7

Pods

Running: 8

Replica Sets

Running: 7

Stateful Sets

Running: 1

Name	Images	Labels	Pods	Created
assurance	fares121/assurance-service:1.0.0	-	1 / 1	a day ago
assurance-policy	fares121/assurancepolicy-service:1.0.0	-	1 / 1	a day ago
cloud-config-server-app	fares121/cloud-config-server:1.0.0	app: cloud-config-server-app	1 / 1	a day ago
cloud-gateway-app	fares121/cloud-gateway:1.0.0	app: cloud-gateway-app	1 / 1	a day ago
department-service-app	fares121/department-service:1.0.0	app: department-service-app	1 / 1	a day ago
postgres	postgres	app: postgres tier: database	1 / 1	a day ago
user-service-app	fares121/user-service:1.0.0	app: user-service-app	1 / 1	a day ago

• Partie préparation du serveur SonarQube :



```
fares@fares-virtual-machine:~$ docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube
```

```
fares@fares-virtual-machine:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
378fbe003ed6 sonarqube "/opt/sonarqube/dock..." 7 days ago Up 16 seconds 0.0.0.0:9000->9000/tcp, :::9000->9000/tcp, 0.0.0.0:9092->9092/tcp, :::9092->9092/tcp
```

The screenshot shows the SonarQube interface with the 'Quality Profiles' tab selected. The main content area displays three sections: 'Recently Added Rules' (with links to 'Disabling Role-Based Access Control on Azure resources is secu...', 'Template evaluation should not expose secure values', 'Don't use "allowedValues" for a location parameter', 'Credentials should not be hard-coded', 'The resource elements should appear in the recommended order', and 'Disabling certificate-based authentication is security-sensitive'), 'AzureResourceManager, 1 profile(s)' (with a table showing 31 rules, updated 7 days ago, and never used), 'Sonar way BUILT-IN' (with a table showing 329 rules, updated 7 days ago, and never used), and 'C#, 1 profile(s)' (with a table showing 24 rules, updated 7 days ago, and never used). A 'Create' button is visible in the top right corner.

sonarqube

Projects Issues Rules Quality Profiles Quality Gates Administration More ?

Quality Profiles

Quality profiles are collections of rules to apply during an analysis. For each language there is a default profile. All projects not explicitly assigned to some other profile will be analyzed with the default. Ideally, all projects will use the same profile for a language. [Learn More](#)

Create Restore

Filter by Select language

Profile	Projects	Rules	Updated	Used	More
AzureResourceManager, 1 profile(s)	Projects	Rules	Updated	Used	
Sonar way BUILT-IN	DEFAULT	31	7 days ago	Never	⋮
C#, 1 profile(s)	Projects	Rules	Updated	Used	
Sonar way BUILT-IN	DEFAULT	329	7 days ago	Never	⋮
CSS, 1 profile(s)	Projects	Rules	Updated	Used	
Sonar way BUILT-IN	DEFAULT	24	7 days ago	Never	⋮

Recently Added Rules

- [Disabling Role-Based Access Control on Azure resources is security-sensitive](#)
AzureResourceManager, activated on 1 profile(s)
- [Template evaluation should not expose secure values](#)
AzureResourceManager, activated on 1 profile(s)
- [Don't use "allowedValues" for a location parameter](#)
AzureResourceManager, activated on 1 profile(s)
- [Credentials should not be hard-coded](#)
AzureResourceManager, activated on 1 profile(s)
- [The resource elements should appear in the recommended order](#)
AzureResourceManager, activated on 1 profile(s)
- [Disabling certificate-based authentication is security-sensitive](#)
AzureResourceManager, activated on 1 profile(s)
- [Elements should not be empty or null](#)
AzureResourceManager, activated on 1 profile(s)
- [Use a hard-coded value for the apiVersion](#)
AzureResourceManager, activated on 1 profile(s)
- [Enabling Azure resource-specific admin accounts is security-sensitive](#)
AzureResourceManager, activated on 1 profile(s)





• Partie Intégration avec Jenkins :

1. Créer le fichier Docker suivant :

Pour cela, nous utiliserons l'image officielle de Jenkins provenant de Docker Hub. Vous pouvez également utiliser vos propres images, assurez-vous simplement d'inclure la commande suivante :

RUN curl -sSL https://get.docker.com/ | sh.

```
1 FROM jenkins/jenkins:2.414.3-jdk17
2
3 USER root
4
5 RUN curl -sSL https://get.docker.com/ | sh
6
7 USER jenkins
```

2. Construire le Dockerfile en une image.

```
fares@fares-virtual-machine:~$ docker build --tag jenkins/jenkins:2.414.3-jdk17 .■
```

3.Trouver l'UID du groupe Docker sur votre système hôte.

C'est le groupe utilisateur de votre installation Docker, et cela est nécessaire pour se connecter à docker.sock de l'hôte.

```
fares@fares-virtual-machine:~$ getent group docker
docker:x:999:jenkins,fares
■
```

- --group-add 999

Cela ajoutera le groupe du système hôte à l'utilisateur Jenkins. Si vous utilisez l'image officielle de Jenkins, sinon, l'utilisateur principal à l'intérieur du conteneur, qui est généralement non-root (par exemple, jenkins, appuser ou similaire), est celui qui obtient ces droits de groupe supplémentaires.

- --volume /var/run/docker.sock:/var/run/docker.sock

Cela liera le conteneur au socket UNIX de Docker de l'hôte, afin que le CLI que nous avons installé à l'intérieur du conteneur puisse utiliser le démon Docker de l'hôte.

4.Démarrer un conteneur à partir de l'image que nous avons construite précédemment

```
fares@fares-virtual-machine:~$ docker run --name jenkins --restart=on-failure --detach --publish 8080:8080 --volume /var/run/docker.sock:/var/run/docker.sock --group-add 999 --volume jenkins_data:/var/jenkins_home jenkins/jenkins:2.41.3-jdk17
```

5. Tester si la configuration a fonctionné :

- Tapez docker ps pour obtenir l'ID du conteneur du conteneur Jenkins:

```
fares@fares-virtual-machine: $ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS
378fbe003ed6 sonarqube "/opt/sonarqube/dock..." 7 days ago Up 22 minutes 0.0.0.0:9000->9000/tcp, :::9000->9000/tcp, f4297cf1cf71 jenkins/jenkins:2.41.3-jdk17 "/usr/bin/tini -- /u..." 9 days ago Up 3 hours 0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 50000/tcp 40ababla6003 gcr.io/k8s-minikube/kicbase:v0.0.44 "/usr/local/bin/entr..." 3 weeks ago Up 3 hours 127.0.0.1:32768->22/tcp, 127.0.0.1:32769->2376/tcp, 127.0.0.1:32770->5000/tcp, 127.0.0.1:32771->8443/tcp, 127.0.0.1:32772->32443/tcp minikube
fares@fares-virtual-machine: $
```

- Exécutez docker exec -it <containerId> bash.

```
fares@fares-virtual-machine:~$ docker exec -it f4297cf1cf71 bash
jenkins@f4297cf1cf71:/$
```

- Exécutez docker build . à l'intérieur du conteneur.

```
jenkins@f4297cf1cf71:~$ docker build .
[+] Building 0.2s (1/1) FINISHED
--> [internal] load build definition from Dockerfile
--> => transferring dockerfile: 2B
ERROR: failed to solve: failed to read dockerfile: open Dockerfile: no such file or directory
jenkins@f4297cf1cf71:/$
```

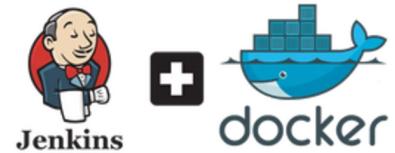
Vous devriez maintenant recevoir le message suivant :

ERROR: failed to solve: failed to read dockerfile

Note sur Jenkins et Docker

Pour pouvoir utiliser le démon Docker et exécuter des commandes Docker dans un pipeline Jenkins en cours d'exécution dans un conteneur, il est nécessaire d'inclure le volume Docker lors de l'exécution de l'image Jenkins créée avec le Dockerfile. De plus, il faut installer le plugin Docker pour Pipeline. Toutes ces étapes sont essentielles pour lier Docker avec Jenkins.

Concernant la deuxième solution, elle est moins simple. Lorsque l'on télécharge Jenkins et qu'on l'extracte dans notre VM, il suffit d'ajouter l'utilisateur Jenkins au groupe Docker pour permettre la communication avec Docker. Ensuite, il est nécessaire d'installer le plugin Docker dans Jenkins pour exécuter des commandes Docker dans vos scripts.





• Partie Intégration avec Minikube:

D'abord, il faut apporter des modifications dans le conteneur Jenkins pour que celui-ci puisse accéder à Minikube. Dans le fichier .kube/config, se trouve la configuration de Minikube, où l'on trouve l'adresse IP du serveur Minikube. Voici les étapes à suivre :

1. Connecter le conteneur Jenkins au même réseau que Minikube : Cela permettra une communication fluide entre les deux .
2. Inclure le volume de kubectl existant dans la VM dans le conteneur : Cela est nécessaire pour pouvoir utiliser la commande kubectl et accéder au serveur AP
3. Lancer un nouveau conteneur Jenkins : Après avoir effectué les modifications nécessaires.

```
fares@fares-virtual-machine:~$ docker network ls
NETWORK ID      NAME          DRIVER      SCOPE
b3cd413bef23   bridge        bridge      local
bf96e3f64959   devops_sonar-net  bridge      local
56f5f149f708   devops_sonarnet  bridge      local
43bbe2a554d9   host          host       local
e73f9361f4e5   minikube      bridge      local
4507de426b11   none          null       local
cf45771a6400   sonarqube_network  bridge      local
fares@fares-virtual-machine:~$
```

```
fares@fares-virtual-machine:~$ docker run --name jenkins --restart=on-failure
--detach --publish 8080:8080 --volume /var/run/docker.sock:/var/run/docker.sock
--group-add 999 --volume jenkins_data:/var/jenkins_home --volume /usr/local/bin/kubectl:/usr/local/bin/kubectl --network minikube jenkins:jenkins:2.414.3-jdk17
```

4. Tester la connectivité avec Minikube : Accédez au tableau de bord de Jenkins et configurez un cloud nommé "Kubernetes". Pointez sur le fichier .kube/config situé dans votre VM (en local). Ensuite, testez si Jenkins peut accéder au serveur Minikube.

```
fares@fares-virtual-machine:~$ cd .kube
fares@fares-virtual-machine:~/ kube$ ls
cache  config
fares@fares-virtual-machine:~/ kube$ █
```

```

fares@fares-virtual-machine:~/kube$ cat config
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /home/fares/.minikube/ca.crt
    extensions:
    - extension:
        last-update: Tue, 03 Sep 2024 22:14:43 CET
        provider: minikube.sigs.k8s.io
        version: v1.33.1
        name: cluster_info
        server: https://[REDACTED]
    name: minikube
contexts:
- context:
    cluster: minikube
    extensions:
    - extension:
        last-update: Tue, 03 Sep 2024 22:14:43 CET
        provider: minikube.sigs.k8s.io
        version: v1.33.1
        name: context_info
    namespace: default
    user: minikube
    name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /home/fares/.minikube/profiles/minikube/client.crt
    client-key: /home/fares/.minikube/profiles/minikube/client.key
fares@fares-virtual-machine:~/kube$ █

```

Cloud kubernetes Configuration

Name ?

Kubernetes Cloud details ^ Edited

Kubernetes URL ?

Use Jenkins Proxy ?

Kubernetes server certificate key ?

Disable https certificate check ?

Kubernetes Namespace

JNLP Docker Registry ?

Inject restricted PSS security context in JNLP container definition ?

Credentials

config (minikube kubeconfig)

Add ▾

Test Connection

Credentials

config (minikube kubeconfig)

Add ▾

Connected to Kubernetes v1.30.0

Test Connection

5. Ajouter les plugins nécessaires de Kubernetes dans Jenkins

Kubernetes

Dashboard > Manage Jenkins > Plugins

Advanced settings

Plugins

Search bar: kub

Name	Enabled
Kubernetes CLI Plugin 1.12.1 Configure kubectl for Kubernetes Report an issue with this plugin	<input checked="" type="checkbox"/> <input type="button" value="X"/>
Kubernetes Client API Plugin 6.10.0-240.v57880ce8b_0b_2 Kubernetes Client API plugin for use by other Jenkins plugins. Report an issue with this plugin	<input checked="" type="checkbox"/> <input type="button" value="X"/>
Kubernetes Credentials Plugin 174.va_36e093562d9 Common classes for Kubernetes credentials Report an issue with this plugin	<input checked="" type="checkbox"/> <input type="button" value="X"/>
Kubernetes Credentials Provider 1.262.v2670ef7ea_0c5 Provides a read only credentials store backed by Kubernetes. Report an issue with this plugin	<input checked="" type="checkbox"/> <input type="button" value="X"/>
Kubernetes plugin 4054.v2da_8e2794884 This plugin integrates Jenkins with Kubernetes Report an issue with this plugin	<input checked="" type="checkbox"/> <input type="button" value="X"/>

REST API Jenkins 2.414.3

Maven pour la création des jars

Plugins

Search bar: maven

Name

Enabled

Maven Integration plugin 3.23

This plugin provides a deep integration between Jenkins and Maven. It adds support for automatic triggers between projects depending on SNAPSHOTs as well as the automated configuration of various Jenkins publishers such as Junit.

[Report an issue with this plugin](#)

Github

Plugins

github

Name	Enabled
GitHub API Plugin 1.321-468.v6a_9f5f2d5a_7e	
This plugin provides GitHub API for other plugins.	
Report an issue with this plugin	
GitHub Branch Source Plugin 1793.v1831e9c68d77	
Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc.	
Report an issue with this plugin	
GitHub Integration Plugin 0.7.0	
GitHub Integration Plugin for Jenkins	
GitHub plugin 1.40.0	
This plugin integrates GitHub to Jenkins.	
Report an issue with this plugin	
Pipeline: GitHub Groovy Libraries 61.v629f2cc41d83	
Allows Pipeline Groovy libraries to be loaded on the fly from GitHub.	
Report an issue with this plugin	

Docker

docke

Name	Enabled
Docker API Plugin 3.3.6-90.ve7c5c7535ddd	
This plugin provides docker-java API for other plugins.	
Report an issue with this plugin	
This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.	
Docker Commons Plugin 443.v921729d5611d	
Provides the common shared functionality for various Docker-related plugins.	
Report an issue with this plugin	
Docker Pipeline 580.vc0c340686b_54	
Build and use Docker containers from pipelines.	
Report an issue with this plugin	
Docker plugin 1.5	
This plugin integrates Jenkins with Docker	
Report an issue with this plugin	

Sonarqube

Plugins

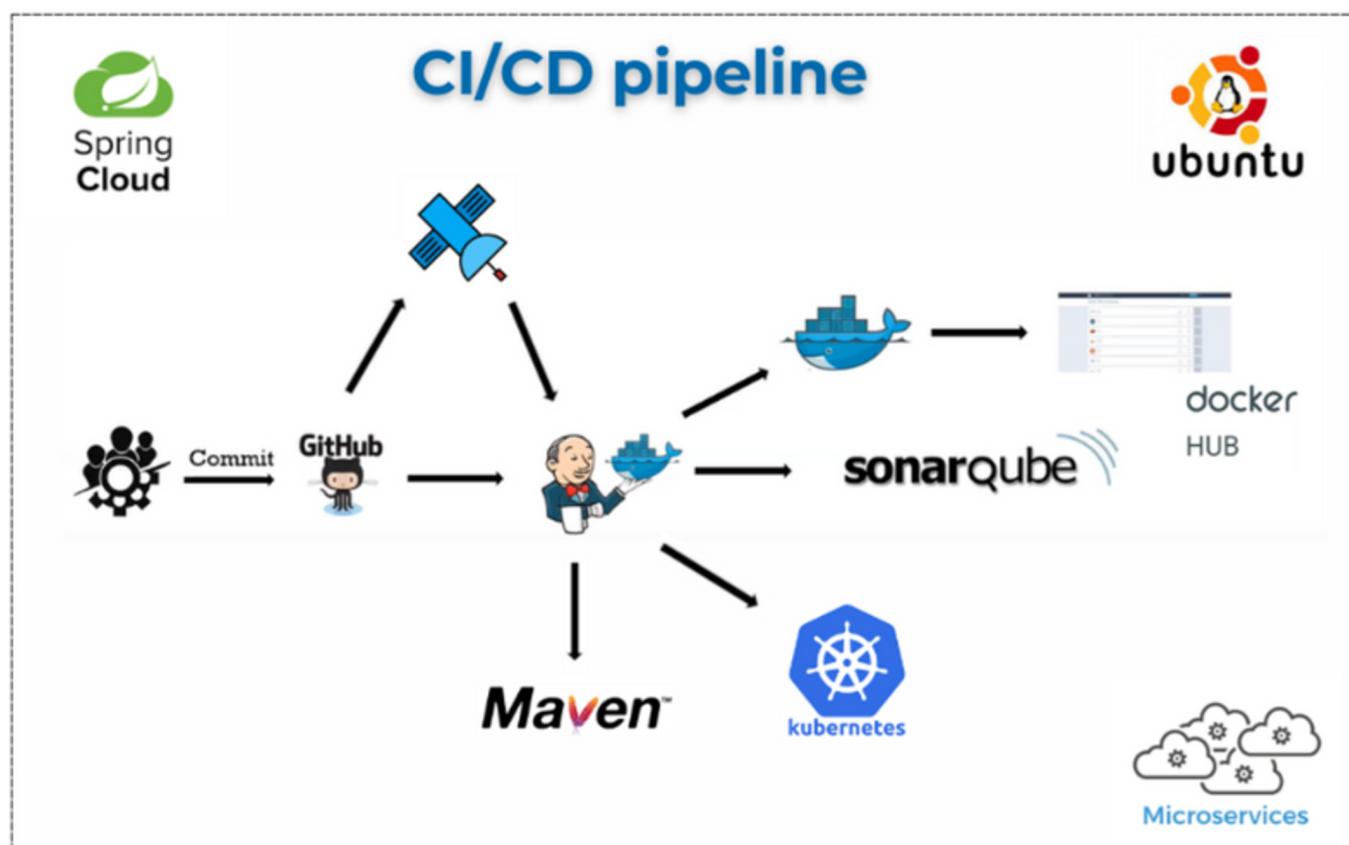
sonar

Name	Enabled
SonarQube Scanner for Jenkins 2.17.2	
This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	
Report an issue with this plugin	

• Partie Automatisation

Dans cette partie, nous allons automatiser toutes les tâches manuelles que j'ai effectuées dans les sections précédentes. Une fois que nous avons configuré tous les outils, les plugins et la connectivité avec les différents serveurs via Jenkins, nous allons créer un pipeline responsable de cette automatisation.

Au final, lorsque Jenkins détecte un commit et un push, il exécute automatiquement le script qui commence par créer le JAR de chaque service. Ce JAR est ensuite envoyé au serveur SonarQube pour l'inspection du code. Par la suite, le script se connecte à Docker Hub, crée l'image et l'envoie sur ce registre. Enfin, le script se connecte à un autre référentiel GitHub où se trouvent les fichiers Kubernetes (déploiement et service) et démarre tous les pods, services et déploiements dans mon environnement local via Jenkins.



Les étapes pour créer ce pipeline :

- Naviguez vers la partie Dashboard dans Jenkins, puis cliquez sur New Item et ajoutez un Pipeline en spécifiant le nom.
- Étant donné que j'ai mon répertoire sur GitHub, je dois choisir l'option GitHub, comme le montre la figure.

The screenshot shows the Jenkins Pipeline configuration interface. At the top, it says "Dashboard > testingMinikube > Configuration".

Build Triggers

Checkboxes for build triggers are shown:

- Build after other projects are built
- Build periodically
- GitHub Branches
- GitHub Pull Requests
- GitHub hook trigger for GITScm polling
- Poll SCM

A note below says: "⚠ Do you really mean "every minute" when you say "*****"? Perhaps you meant "H * * * *" to poll once per hour". It also indicates the last run was on Tuesday, September 3, 2024 at 9:25:52 PM Coordinated Universal Time, and the next run is scheduled for the same time.

Other trigger options shown are:

- Ignore post-commit hooks
- Quiet period
- Trigger builds remotely (e.g., from scripts)

Advanced Project Options

An "Advanced" dropdown is open, showing the same trigger options again.

Definition

The "Definition" section is set to "Pipeline script from SCM".

SCM

The "SCM" dropdown is set to "Git".

Repositories

The "Repository URL" field contains the URL: <https://github.com/benammarfares/K8s-Microservice-SpringCloud.git>.

- J'ai spécifié le nom de la branche où se trouve le service et inclus le Jenkinsfile, qui sera responsable de l'automatisation du pipeline, dans le projet GitHub.

The screenshot shows the Jenkins Pipeline configuration interface. Under 'Branches to build', the 'Branch Specifier' is set to '*/main'. In the 'Script Path' section, 'Jenkinsfile' is specified. The 'Lightweight checkout' option is checked. At the bottom, there are 'Save' and 'Apply' buttons.

- Dans mon Jenkinsfile, j'ai créé un JAR pour chaque service que je possède et envoyé la structure du code ainsi que le code lui-même au serveur SonarQube, qui va le scanner de la manière illustrée dans la figure.
- Ce processus est effectué pour tous les services.
- Dans le stage ci-dessous, nous avons la création de l'image Docker à partir du JAR, ainsi que la connexion à Docker Hub et l'envoi de l'image avec son tag.

```

stage('Build Config Server') {
    agent any
    steps {
        checkout scm
        script {
            sh "pwd"
            sh "ls -la"
            dir("cloud-config-server") {
                sh 'mvn compiler:compile'
                withSonarQubeEnv('sonarserver') {
                    sh 'mvn sonar:sonar'
                }
                sh 'mvn clean package -DskipTests'
            }
        }
    }
}

stage('Push Config Docker Image') {
    steps {
        script {
            sh "pwd"
            sh "ls -la"
            dir("${CUSTOM_WORKSPACE}/cloud-config-server") {
                sh 'docker build -t fares121/cloud-config-server:1.0.0 .'
                withCredentials([string(credentialsId: 'Docker', variable: 'docker_password')]) {
                    sh 'docker login -u fares121 -p ${docker_password}'
                    sh 'docker push fares121/cloud-config-server:1.0.0'
                }
            }
        }
    }
}

```

```

stage('Checkout to the kubernetes github repo') {
    steps {
        cleanWs()
        git branch: 'main',
            url: 'https://github.com/benammarfares/Kubernetes-yml-for-service-deploy.git'
        sh 'ls -la'
        sh 'git rev-parse HEAD'
        withKubeConfig(credentialsId: 'mykubeconfig') {
            sh 'kubectl get all'
            sh 'kubectl apply -f ./'
            sh 'kubectl get all'
        }
    }
}

stage('Exposing Eureka Cloud Service') {
    steps {
        withKubeConfig(credentialsId: 'mykubeconfig') {
            sh ''
            kubectl port-forward service/eureka-lb 31744:80 &
            echo $! > eureka-lb.pid
            ...
            sh 'sleep 5'
        }
    }
}

stage('Exposing Gateway') {
    steps {
        withKubeConfig(credentialsId: 'mykubeconfig') {
            sh ''
            kubectl port-forward service/cloud-gateway-svc 30950:80 &
            echo $! > cloud-gateway-svc.pid
            ...
            sh 'sleep 5'
        }
    }
}

```

- Dans ce stage, j'ai pointé vers un autre répertoire GitHub où se trouvent mes fichiers Kubernetes pour lancer les services et les déploiements.
- D'abord, j'ai inspecté tous les services et déploiements avec la commande `kubectl get all`, puis j'ai exécuté les fichiers Kubernetes et inspecté de nouveau pour voir les nouveaux services et déploiements.
- Ensuite, dans le deuxième stage, j'ai exposé le serveur Eureka.
- Le stage final a été consacré à l'exposition de la passerelle (LoadBalancer).

Created	Access Key	Tunnel Whitelist	API Access	Description	Last used
Aug 26th 2024, 11:30:19 am	616e78f3-8872-47c9-bd7e-78584de38a7f				1 week ago

Rows per page: 10 1-1 of 1 < >

The screenshot shows the GitHub 'Webhooks / Manage webhook' page for the repository 'benammarfares/K8s-Microservice-SpringCloud'. The left sidebar contains navigation links for General, Access, Collaborators, Moderation options, Code and automation (Branches, Tags, Rules, Actions), Webhooks (selected), Environments, Codespaces, Pages, Security (Code security and analysis, Deploy keys, Secrets and variables), Integrations (GitHub Apps, Email notifications, Autolink references), and SSL verification.

The main content area displays the configuration for a webhook:

- Payload URL ***: https://hooks.webhookrelay.com
- Content type ***: application/x-www-form-urlencoded
- Secret**: (empty field)
- SSL verification**: By default, we verify SSL certificates when delivering payloads. Options: Enable SSL verification (selected) or Disable (not recommended).
- Which events would you like to trigger this webhook?** Options: Just the push event (selected), Send me everything, or Let me select individual events.
- Active**: A checked checkbox with the note: We will deliver event details when this hook is triggered.

At the bottom are 'Update webhook' and 'Delete webhook' buttons.

- Ici, j'ai accédé au site web de WebhookRelay et créé un token. Ensuite, via le terminal, je me suis connecté au relay CLI en utilisant les identifiants comme variables d'environnement. J'ai également configuré le webhook du répertoire GitHub avec l'URL générée par le relay.
- Maintenant, le pipeline peut détecter, à travers le webhook, chaque commit et push dans le répertoire GitHub et exécuter automatiquement le pipeline.

Add description

Started 8 days 2 hr ago
Took 41 minStarted by user [fares](#)

This run spent:

- 26 ms waiting;
- 41 min build duration;
- 41 min total from scheduled to completion.



Revision: ba90dd31db191d331979729892696c27adacf9fd

Repository: <https://github.com/benammarfares/K8s-Microservice-SpringCloud.git>

- refs/remotes/origin/main



Revision: bf6d9dbe811c6ed51c2a4a739e9b8e18b5d7821d

Repository: <https://github.com/benammarfares/Kubernetes-yml-for-service-deploy.git>

- refs/remotes/origin/main



kubernetes default Search

Workloads

Workloads (Running: 7) Deployments (Running: 8) Pods (Running: 7) Replica Sets (Running: 1) Stateful Sets

Deployments					
Name	Images	Labels	Pods	Created	⋮
assurance	fares121/assurance-service:1.0.0	-	1 / 1	a day ago	⋮
assurance-policy	fares121/assurancepolicy-service:1.0.0	-	1 / 1	a day ago	⋮
cloud-config-server-app	fares121/cloud-config-server:1.0.0	app: cloud-config-server-app	1 / 1	a day ago	⋮
cloud-gateway-app	fares121/cloud-gateway:1.0.0	app: cloud-gateway-app	1 / 1	a day ago	⋮
department-service-app	fares121/department-service:1.0.0	app: department-service-app	1 / 1	a day ago	⋮
postgres	postgres	app: postgres tier: database	1 / 1	a day ago	⋮
user-service-app	fares121/user-service:1.0.0	app: user-service-app	1 / 1	a day ago	⋮

127.0.0.1:45743/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/cluster?namespace=default

Le pipeline est maintenant en état de succès, et les pods pour la passerelle et le serveur Eureka sont exposés. Tous les services ont pu s'enregistrer auprès du serveur Eureka, et toutes les API peuvent être envoyées via la passerelle, qui joue le rôle de LoadBalancer.

faresops

Activities Terminal

fares@fares-virtual-machine: ~

```

NAME          READY   STATUS    RESTARTS   AGE
pod/assurance-855c6cc489-rmmt8   1/1    Running   4 (8m53s ago)   38h
pod/assurance-policy-84748ddf9c-flnkf  1/1    Running   4 (11h ago)    38h
pod/cloud-config-server-app-f95db9898-jt8s9  1/1    Running   4 (11h ago)    38h
pod/cloud-gateway-app-5fb64b6446-fj5xt  1/1    Running   4 (11h ago)    38h
pod/department-service-app-5f898bd5fb-2c4mk  1/1    Running   4 (8m53s ago)   38h
pod/eureka-0           1/1    Running   4 (11h ago)    38h
pod/postgres-78d885c78d-6ur58   1/1    Running   4 (11h ago)    38h
pod/user-service-app-57dbc6fcfc6-rcqn8  1/1    Running   4 (11h ago)    38h

NAME          TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)  AGE
service/assurance  ClusterIP   <none>        80/TCP     38h
service/assurance-policy ClusterIP   <none>        80/TCP     38h
service/cloud-config-server-svc ClusterIP   <none>        80/TCP     38h
service/cloud-gateway-svc LoadBalancer <pending>    80:30950/TCP 38h
service/department-service-svc ClusterIP   <none>        80/TCP     38h
service/eureka       ClusterIP   <none>        8761/TCP   38h
service/eureka-lb    NodePort    <none>        80:31744/TCP 38h
service/kubernetes  ClusterIP   <none>        443/TCP    17d
service/postgres    ClusterIP   <none>        5432/TCP   38h
service/user-service-svc ClusterIP   <none>        80/TCP     38h

NAME          READY   UP-TO-DATE  AVAILABLE  AGE
deployment.apps/assurance  1/1     1           1          38h
deployment.apps/assurance-policy  1/1     1           1          38h
deployment.apps/cloud-config-server-app  1/1     1           1          38h
deployment.apps/cloud-gateway-app  1/1     1           1          38h
deployment.apps/department-service-app  1/1     1           1          38h
deployment.apps/postgres   1/1     1           1          38h
deployment.apps/user-service-app  1/1     1           1          38h

NAME          DESIRED  CURRENT  READY  AGE
replicaset.apps/assurance-855c6cc489  1        1        1    38h
replicaset.apps/assurance-policy-84748ddf9c  1        1        1    38h
replicaset.apps/cloud-config-server-app-f95db9898  1        1        1    38h
replicaset.apps/cloud-gateway-app-5fb64b6446  1        1        1    38h
replicaset.apps/department-service-app-5f898bd5fb  1        1        1    38h
replicaset.apps/postgres-78d885c78d  1        1        1    38h
replicaset.apps/user-service-app-57dbc6fcfc6  1        1        1    38h

NAME          READY   AGE

```

Stage View

Delete Pipeline

Full Stage View

SonarQube

SonarQube

SonarQube

SonarQube

SonarQube

SonarQube

Rename

Pipeline Syntax

Github Hook Log

Git Polling Log

Build History

Filter builds...

Aug 26, 2024, 7:02 PM

Aug 26, 2024, 6:53 PM

Aug 26, 2024, 6:53 PM

Average stage times:
(Average full run time: ~41min 23s)

Decla Tool

Search for projects... Perspective Overall Status Sort by Name 5.1 7 project(s) Passed

assurance PUBLIC
Last analysis: 12 hours ago - 269 Lines of Code - Java, XML
A 0 Security A 0 Reliability A 0 Maintainability A — Hotspots Reviewed 0.0% Coverage 0.0% Duplications Passed

assurancePolicy PUBLIC
Last analysis: 12 hours ago - 224 Lines of Code - Java, XML
A 0 Security A 1 Reliability A 8 Maintainability A — Hotspots Reviewed 0.0% Coverage 0.0% Duplications Passed

cloud-config-server PUBLIC
Last analysis: 13 hours ago - 76 Lines of Code - XML, Java
A 0 Security A 0 Reliability A 0 Maintainability A — Hotspots Reviewed 0.0% Coverage 0.0% Duplications Passed

cloud-gateway PUBLIC
Last analysis: 12 hours ago - 98 Lines of Code - XML, Java
A 0 Security A 0 Reliability A 0 Maintainability A — Hotspots Reviewed 0.0% Coverage 0.0% Duplications Passed

department-service PUBLIC
Last analysis: 12 hours ago - 169 Lines of Code - XML, Java
A 0 Security A 0 Reliability A 0 Maintainability A — Hotspots Reviewed 0.0% Coverage 0.0% Duplications Passed

user-service PUBLIC
Last analysis: 12 hours ago - 169 Lines of Code - XML, Java
A 0 Security A 0 Reliability A 0 Maintainability A — Hotspots Reviewed 0.0% Coverage 0.0% Duplications Passed

SonarQube Quality Gate

cloud-config-server	Passed
server-side processing:	Success
service-registry	Passed
server-side processing:	Success
cloud-gateway	Passed
server-side processing:	Success
assurance	Passed
server-side processing:	Success
assurancePolicy	Passed
server-side processing:	Success
department-service	Passed
server-side processing:	Success
user-service	Passed
server-side processing:	Success

- Partie Mise à l'échelle des services :

spec:

```
selector:
  matchLabels:
    app: assurance
replicas: 3
```

spec:

```
replicas: 2
selector:
  matchLabels:
    app: department-service-app
```

Concernant la mise à l'échelle, j'ai modifié le nombre de réplicas dans le déploiement des services. Cela me permettra de résoudre les problèmes de panne qui peuvent survenir lorsqu'un pod spécifique subit une surcharge de trafic.

Kubernetes propose la solution des réplicas : lorsqu'un service tombe, il est remplacé par une instance similaire, assurant ainsi la continuité du service, comme le montrent les deux figures.

NAME	READY	STATUS	RESTARTS	AGE
pod/assurance-76b69b56cf-2trvp	1/1	Running	0	7s
pod/assurance-76b69b56cf-gsjlf	1/1	Running	0	7s
pod/assurance-76b69b56cf-qp8wp	1/1	Running	0	7s
pod/assurance-policy-55b7bd9475-jk29k	1/1	Running	0	7s
pod/assurance-policy-55b7bd9475-jtbhl	1/1	Running	0	7s
pod/assurance-policy-55b7bd9475-jxf9j	1/1	Running	0	7s
pod/cloud-config-server-app-7f7855d956-n8bcp	1/1	Running	0	7s
pod/cloud-gateway-app-857f57554d-kb42q	1/1	Running	0	7s
pod/department-service-app-8c64bf8db-g4w82	1/1	Running	0	7s
pod/department-service-app-8c64bf8db-h4cx7	1/1	Running	0	7s
pod/eureka-0	1/1	Running	0	7s
pod/postgres-78d885c78d-fg686	1/1	Running	0	7s
pod/user-service-app-579b97b874-cv65c	1/1	Running	0	7s
pod/user-service-app-579b97b874-hbbfq	1/1	Running	0	7s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/assurance	3/3	3	3	8s
deployment.apps/assurance-policy	3/3	3	3	7s
deployment.apps/cloud-config-server-app	1/1	1	1	7s
deployment.apps/cloud-gateway-app	1/1	1	1	7s
deployment.apps/department-service-app	2/2	2	2	7s
deployment.apps/postgres	1/1	1	1	7s
deployment.apps/user-service-app	2/2	2	2	7s

NAME	DESIRED	CURRENT	READY
replicaset.apps/assurance-76b69b56cf	3	3	3
replicaset.apps/assurance-policy-55b7bd9475	3	3	3
replicaset.apps/cloud-config-server-app-7f7855d956	1	1	1
replicaset.apps/cloud-gateway-app-857f57554d	1	1	1
replicaset.apps/department-service-app-8c64bf8db	2	2	2
replicaset.apps/postgres-78d885c78d	1	1	1
replicaset.apps/user-service-app-579b97b874	2	2	2

Conclusion

Grâce à ce processus d'automatisation, j'ai pu réduire le temps consacré à la gestion des différentes phases à travers les étapes du pipeline, qui sont déclenchées automatiquement grâce à Jenkins et WebhookRelay.

En ce qui concerne l'analyse de code, SonarQube s'est révélé très efficace, me fournissant une interface facile à comprendre et à gérer pour consulter et inspecter la structure et la syntaxe de mes services.

D'autre part, le déploiement local et l'exposition des microservices basés sur Spring Cloud ont été un succès, grâce à l'utilisation de Minikube et de Postman pour les tests.