ESSENTIELS DE L'INFRASTRUCTURE DU SI PARTIE 1

RAPPEL SUR LES INFRASTRUCTURES

Infrastructure Traditionnelle :

Dans les infrastructures traditionnelles, les ressources informatiques telles que les serveurs, le stockage et le réseau sont physiquement présents dans le data center de l'entreprise. Ils nécessitent un investissement initial important en matériel et en maintenance, et la capacité doit être planifiée à l'avance pour éviter les pénuries ou le surdimensionnement.

Le Cloud :

Le modèle cloud, quant à lui, s'appuie sur Internet pour fournir les mêmes types de ressources mais de manière virtualisée et à la demande. Il offre une flexibilité et une scalabilité inégalées, où les ressources peuvent être ajustées en fonction des besoins actuels, souvent avec un modèle de paiement à l'utilisation. Mais cela fait aussi apparaître la notion de dépendance.

Comparaison :

La transition vers le cloud est motivée par plusieurs avantages clés sur l'infrastructure traditionnelle :

- **Élasticité** : Capacité à augmenter ou réduire les ressources rapidement.
- **Résilience** : Meilleure tolérance aux pannes grâce à des services redondants.
- **Maintenance** : Diminution de la charge de travail sur la gestion des équipements.
- **Accessibilité** : Accès global facilité par l'Internet.

VM ET CONTENEUR

Les VMs et les conteneurs sont deux technologies de virtualisation qui permettent de faire fonctionner les applications de manière isolée et efficace.

VMs :

Une machine virtuelle est une émulation d'un ordinateur qui exécute un système d'exploitation complet avec ses propres ressources virtuelles (CPU, mémoire, disque, interfaces réseau). Chaque VM est isolée des autres, ce qui signifie que les applications peuvent être exécutées dans des environnements différents sur le même hôte physique.

Conteneurs :

Les conteneurs, en revanche, virtualisent l'OS plutôt que le matériel. Cela les rend beaucoup plus légers et rapides à démarrer que les VMs. Docker est l'un des outils de conteneurisation les plus populaires et il permet de packager une application et ses dépendances dans un conteneur qui peut être exécuté sur n'importe quel système d'exploitation Linux.

Comparaison Docker et VM :

Fonctionnalité	Docker	VM
Heure de démarrage	Démarre dans quelques secondes	Cela prend quelques minutes pour démarrer
Taille	Léger (Mo)	Lourd (GB)
Isolement	Partage le système d'exploitation hôte	Système d'exploitation complet pour chaque machine virtuelle
Utilisation des ressources	Efficace, faible coût	Nécessite plus de ressources
Portabilité	Facile à déplacer entre les environnements	Plus difficile à migrer

En termes simples, une VM est comme un appartement entièrement meublé ; elle comprend tout, des murs aux meubles (le système d'exploitation complet et l'application) tandis que le conteneur Docker est comme une simple pièce meublée ; il est petit, léger et contient uniquement ce qui est nécessaire aux applications ainsi que les ressources communes partagées (le système d'exploitation) avec d'autres conteneurs.

DOCKER

Au fil du temps, les développeurs recherchent des moyens plus simples de gérer et de déployer des applications. Et l'un de ces outils qui a gagné beaucoup de terrain au cours des dernières années est Docker.

Qu'est-ce que Docker ?

Docker est une plateforme qui permet aux développeurs de facilement emballer, expédier et exécuter n'importe quelle application sous forme de conteneur léger. Les conteneurs sont entièrement transportables et peuvent être exécutés de manière cohérente sur n'importe quelle machine configurée avec Docker, qu'il s'agisse d'une machine locale ou de machines virtuelles cloud.

Considérez Docker comme un conteneur pour votre code. Tout comme les conteneurs d'expédition peuvent expédier des marchandises dans le monde entier dans un conteneur standard, les conteneurs Docker vous permettent d'exécuter votre application dans n'importe quel conteneur sans vous soucier de ce qu'il contient.

Pourquoi Docker est utile ?

Docker résout de nombreux problèmes courants auxquels les développeurs sont confrontés :

Cohérence entre les environnements : Supposons que vous ayez développé une application sur votre machine locale et que tout fonctionne parfaitement, mais que lorsque vous l'avez déployée sur un serveur, tout se soit cassé. Docker fournit une solution à ce problème, il crée un environnement standardisé (conteneur) dans lequel votre application fonctionnera exactement de la même manière, quel que soit l'endroit où vous la déployez.

Isolation : chaque conteneur de Docker est isolé des autres conteneurs, ce qui signifie que si vous exécutez plusieurs services (par exemple, une base de données et une application Web), ils n'interféreront pas les uns avec les autres.

Léger et rapide : les conteneurs Docker sont légers par rapport aux machines virtuelles (VM) traditionnelles car ils partagent le système d'exploitation hôte, ils démarrent donc rapidement et utilisent moins de ressources.

Portabilité : Comme les conteneurs Docker contiennent tout ce qui est nécessaire pour exécuter une application, vous pouvez facilement les déplacer entre différents environnements, c'est-à-dire de votre ordinateur portable vers un serveur sur le cloud sans aucun problème.

Comment fonctionne Docker ?

Docker utilise une architecture client-serveur :

Le client Docker est l'élément avec lequel vous, le développeur, interagissez (par exemple, en utilisant des commandes telles que docker run).

Le démon Docker (serveur) effectue le gros du travail en gérant les conteneurs.

Les images Docker sont des plans pour les conteneurs. Lorsque vous créez un conteneur Docker, vous créez en fait une instance en cours d'exécution d'une image Docker.

Concepts clés :

Images Docker : un aperçu de tout ce dont votre application a besoin pour fonctionner. Considérez-le comme un modèle pour les conteneurs. Les images peuvent être créées à partir d'un fichier appelé Dockerfile .

Conteneurs Docker : une instance en cours d'exécution d'une image Docker. Vous pouvez le considérer comme un environnement isolé dans lequel votre application s'exécute.

Docker Hub : un référentiel dans lequel vous pouvez trouver des images Docker prédéfinies. Vous pouvez soit extraire des images existantes de Docker Hub, soit envoyer les vôtres.

L'architecture de Docker comprend donc trois composants principaux :

🔹 Client Docker

C'est l'interface à travers laquelle les utilisateurs interagissent. Il communique avec le démon Docker.

🔹 Hôte Docker

Ici, le démon Docker écoute les requêtes de l'API Docker et gère divers objets Docker, notamment les images, les conteneurs, les réseaux et les volumes.

🔹 Registre Docker

C'est ici que les images Docker sont stockées. Docker Hub, par exemple, est un registre public largement utilisé.

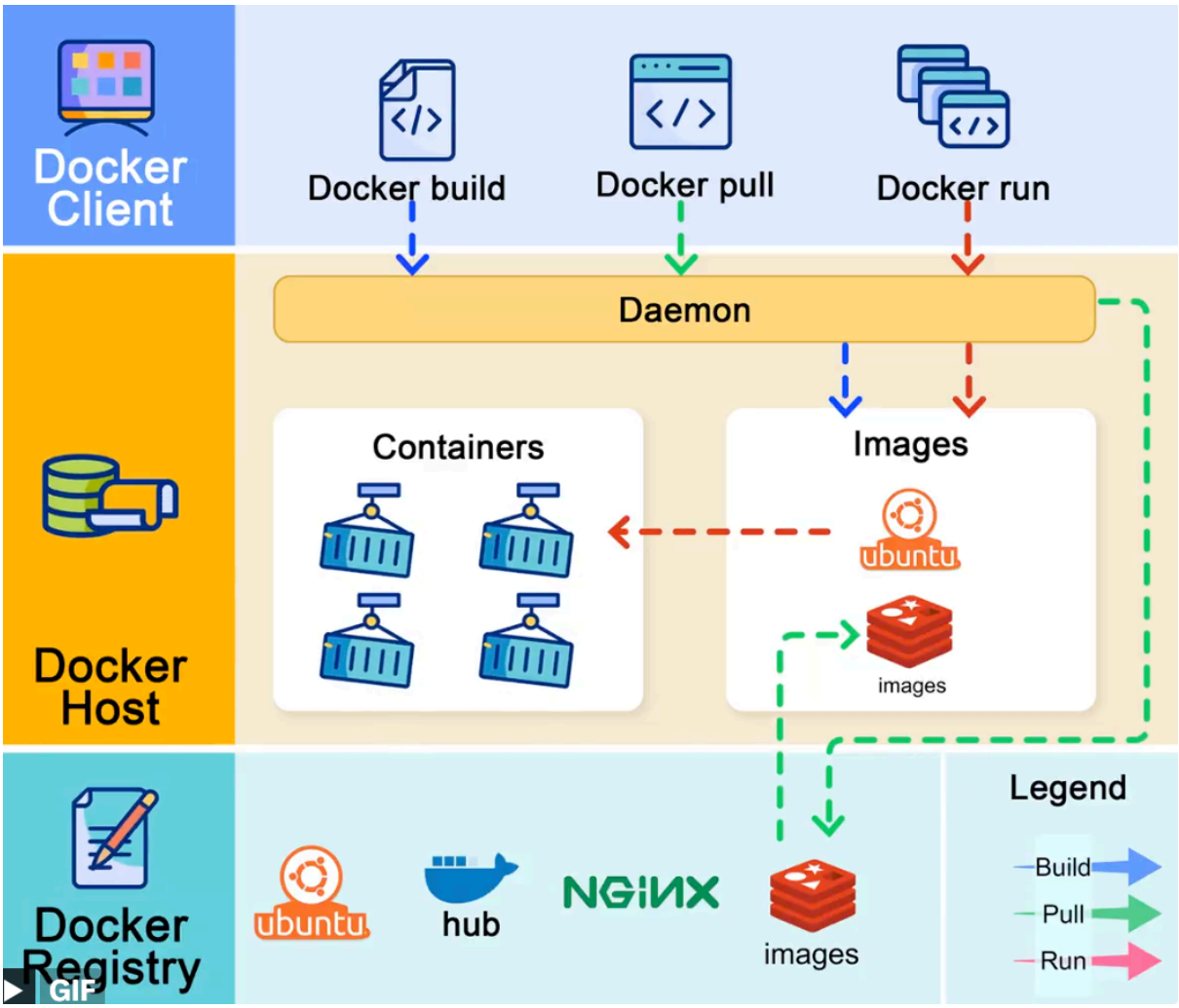
Avantages de Docker

Environnement de développement cohérent : votre application fonctionnera de la même manière partout (local, intermédiaire, production).

Collaboration améliorée : les équipes peuvent partager des images Docker, ce qui permet de collaborer même sur des projets de grande envergure.

Déploiement simplifié : Docker rend le déploiement de vos applications dans différents environnements incroyablement simple et garantit que votre application fonctionnera de la même manière partout.

Efficacité des ressources : les conteneurs sont légers et nécessitent moins de ressources que les machines virtuelles traditionnelles, ce qui en fait un choix idéal pour l'architecture de microservices moderne.



source : https://blog.bytebytego.com/

ATELIER 1 MONGODB EN LOCAL AVEC DOCKER

Dans cet atelier, nous allons découvrir l'intégration de MongoDB dans votre environnement local à l'aide de Docker sur Ubuntu.

1- Avoir l'application Docker installé.

2- Créer un fichier Docker Compose qui se nommera docker-compose.yml.

3- Y ajouter l'image Docker pour Mongo DB :

```yaml
version: "3.9"

services:
    mongo:
        image: mongo:7
        container_name: mongo
        restart: always
        ports:
            - "27017:27017"
        environment:
            MONGO_INITDB_ROOT_USERNAME: root
            MONGO_INITDB_ROOT_PASSWORD: example
        volumes:
            - ./db_data:/data/db
```

- **version**: "3.1": désigne la version du fichier Docker Compose utilisée.
- **services**: Définit les services que nous utilisons dans notre configuration Docker.
- **mongo**: Nom du service pour notre conteneur MongoDB.
- **image: mongo**: Spécifie l'image MongoDB Docker à utiliser.
- **restart: always**: Garantit que le conteneur MongoDB redémarre automatiquement en cas de panne.
- **container_name: mongo**: Attribue un nom spécifique à notre conteneur MongoDB pour une identification facile.
- **ports: - 27017:27017**: Mappe le port 27017 de la machine hote au port 27017 du conteneur MongoDB.
- **environment**: configure le nom d'utilisateur et le mot de passe root initiaux de MongoDB.
- **volumes**: - ./db_data/:/data/db/: Lie un répertoire local (./db_data/) au répertoire de données du conteneur MongoDB.

4- Installer MongoDB sur votre machine en locale à l'aide de Docker et l'image que l'on vient de créer en exécutant la commande suivante :

```
docker compose up -d
```

Cette commande lancera le conteneur MongoDB en mode détaché, vous permettant de continuer à travailler sans interruption.

5- Vérifier l'installation grâce au Shell Mongo

```
docker exec -it mongo mongosh -u root -p example
```

6- Ensuite dans le shell mongo :

```
use myapp
db.createCollection("users")
db.users.insertOne({ name: "Alice" })
show dbs
show collections
```

7- Connexion au conteneur Docker MongoDB :
Pour interagir avec votre conteneur MongoDB, vous devez vous y connecter depuis votre application ou un client MongoDB. Utilisez la chaîne de connexion suivante :

```
mongodb://nom-utilisateur:password@localhost:27017/admin
```

Remplacez nom-utilisateur et password par votre nom d'utilisateur et votre password.
Rappel : une base Mongo n'existe réellement qu'après création d'au moins une collection ou insertion d'un document.

8- Conseils pour une utilisation efficace de MongoDB avec Docker :
- **Persistance des données** : exploitez les volumes Docker pour la persistance des données, garantissant ainsi que vos données survivent aux redémarrages des conteneurs.
- **Personnaliser l'authentification** : modifiez le MONGO_INITDB_ROOT_USERNAMEet MONGO_INITDB_ROOT_PASSWORD dans le docker-compose.yml fichier pour une sécurité renforcée.
- **Mappage des ports** : ajustez le mappage des ports au cas où le port 27017 serait déjà utilisé sur votre machine.

Ressources :

https://docs.docker.com/

https://www.mongodb.com/docs/

https://docs.docker.com/compose/

ATELIER 2 : MISE EN PLACE D'UNE VM ET D'UN CONTENEUR

Objectif de l'Atelier :

Vous mettrez en pratique la théorie vue précédemment en configurant votre propre VM et en déployant un conteneur Docker.

Résultat attendu :

À la fin de cet atelier, vous aurez une VM fonctionnelle et un conteneur exécutant une application web. Nous passerons en revue les étapes ensemble et discuterons des problèmes courants ainsi que des meilleures pratiques.

Étapes pour la VM :

- Installer VirtualBox sur votre machine hôte.
- Créer une nouvelle VM, attribuer des ressources (comme le CPU, la mémoire, et le stockage).
- Installer un système d'exploitation invité, tel que Ubuntu Server.

Étapes pour Docker :

- Installer Docker sur votre machine hôte (ou la VM que vous venez de créer).
- Télécharger une image Docker (comme une image Nginx ou Apache).
- Lancer un conteneur à partir de cette image et accéder à l'application via un navigateur web.

Partie 1: Mise en Place d'une Machine Virtuelle avec VirtualBox

Objectif :

Installer et configurer une VM sur VirtualBox avec un système d'exploitation Linux.

Étapes :

1- Installation de VirtualBox :
2- Téléchargez VirtualBox depuis le site officiel.
3- Exécutez le fichier d'installation et suivez les instructions pour installer VirtualBox sur votre système.

Création d'une Nouvelle VM :

- Lancez VirtualBox et cliquez sur "Nouvelle" pour créer une nouvelle machine virtuelle.
- Nommez votre VM (par exemple, "UbuntuDev") et choisissez "Linux" comme type et "Ubuntu (64-bit)" comme version.
- Attribuez au moins 2 GB de mémoire RAM à votre VM dans l'écran suivant.

Configuration du Disque Dur Virtuel :

- Choisissez de créer un nouveau disque dur virtuel maintenant.
- Sélectionnez VDI (VirtualBox Disk Image) comme type de fichier de disque dur.
- Choisissez "Dynamiquement alloué" pour que le disque grandisse au fur et à mesure de l'utilisation.
- Attribuez 20 GB d'espace disque et créez le disque virtuel.

Installation du Système d'Exploitation :

- Téléchargez une image ISO de la distribution Linux de votre choix, comme Ubuntu Server depuis le site d'Ubuntu.
- Dans VirtualBox, sélectionnez votre VM et cliquez sur "Démarrer".
- Lorsqu'il vous sera demandé de sélectionner un disque de démarrage, choisissez l'image ISO que vous avez téléchargée.
- Suivez les instructions à l'écran pour installer le système d'exploitation.

Configuration Réseau :

- Une fois l'installation terminée, configurez le réseau en mode "Pont" pour que votre VM apparaisse comme un autre ordinateur sur votre réseau.

Connexion SSH (optionnel) :

- Installez un serveur SSH sur votre VM si cela n'a pas été fait pendant l'installation.
- Utilisez un client SSH sur votre machine hôte pour vous connecter à votre VM en utilisant son adresse IP.

Partie 2: Mise en Place d'un Conteneur Docker

Objectif :

Installer Docker et exécuter un conteneur avec une application web simple.

Étapes :

Installation de Docker :

- Sur votre VM ou machine hôte, installez Docker en suivant les instructions du site de Docker.
- Sur Ubuntu, cela peut généralement être fait avec les commandes suivantes :

```
sudo apt update
sudo apt install docker.io
```

Lancement d'un Conteneur Docker :

Lancez un conteneur en utilisant une image existante, par exemple Nginx, qui est un serveur web populaire :

```
sudo docker run --name webserver -d -p 80:80 nginx
```

L'option -d indique à Docker de lancer le conteneur en arrière-plan.

-p 80:80 mappe le port 80 de la machine hôte au port 80 du conteneur, permettant ainsi d'accéder au serveur web.

Vérification :

Ouvrez un navigateur web et accédez à l'adresse IP de votre VM ou machine hôte.

Vous devriez voir la page d'accueil par défaut de Nginx, ce qui indique que votre conteneur est opérationnel.

Interaction avec le Conteneur :

Pour interagir avec votre conteneur, utilisez :

```
sudo docker exec -it webserver bash
```

Cela vous permettra d'entrer dans le shell du conteneur où vous pourrez exécuter des commandes Linux.

Partie 3: Configuration Réseau de Base pour les Infrastructures

Objectif :

Configurer une adresse IP statique pour une machine virtuelle.

Étapes :

Accédez au fichier de configuration réseau de votre VM :

Sur Ubuntu, le fichier se trouve généralement à /etc/netplan/.

Utilisez sudo nano /etc/netplan/50-cloud-init.yaml pour éditer le fichier de configuration.

Configurez une adresse IP statique :

Modifiez le fichier pour définir une adresse IP statique, par exemple :

```
network:
  ethernets:
    enp0s3:
      addresses: [192.168.1.100/24]
      gateway4: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
  version: 2
```

Appliquez la configuration :

Enregistrez le fichier et appliquez la configuration avec sudo netplan apply.

Vérifiez que la configuration est correcte en utilisant ip addr.

RÉSEAU ET STOCKAGE : THÉORIE

Les adresses IP :

Les adresses IP (Internet Protocol) sont des identifiants uniques attribués à chaque appareil sur un réseau. Elles sont composées de quatre octets (dans le cas d'IPv4), allant de 0 à 255, séparés par des points (par exemple, 192.168.1.1). Les adresses IP permettent aux appareils de s'identifier et de communiquer entre eux sur un réseau.

Les masques de sous-réseau :

Un masque de sous-réseau détermine la taille du réseau et la taille de la partie hôte d'une adresse IP. Le masque de sous-réseau est souvent présenté sous la forme d'une suite de quatre octets (comme une adresse IP), par exemple, 255.255.255.0. Il peut également être présenté en notation CIDR, comme /24, ce qui indique que les 24 premiers bits de l'adresse IP sont utilisés pour l'identification du réseau.

Les passerelles :

La passerelle par défaut (ou la passerelle) est l'appareil qui sert de point de jonction entre un réseau local et un autre réseau, généralement Internet. Les appareils sur le réseau local enverront tout le trafic destiné à des adresses en dehors de leur réseau local à la passerelle par défaut.

Les DNS :

Le système de noms de domaine (DNS) est un service qui traduit les noms de domaine lisibles par l'homme en adresses IP numériques. Par exemple, lorsque vous tapez www.example.com dans votre navigateur, le DNS traduit ce nom de domaine en une adresse IP à laquelle votre navigateur peut se connecter.

CONNEXION RÉSEAU DANS UNE VM

Dans un environnement virtuel, plusieurs options de réseau permettent de connecter les machines virtuelles entre elles et avec le monde extérieur.

NAT (Network Address Translation)

Le NAT est utilisé pour mapper plusieurs adresses IP privées à une seule adresse IP publique. C'est ce qui permet à plusieurs machines virtuelles de partager la même connexion Internet de l'hôte sans avoir besoin de leurs propres adresses IP publiques. Dans VirtualBox, le mode NAT relie la VM à Internet à travers l'adresse IP de l'hôte, mais les autres appareils sur le même réseau ne peuvent pas accéder directement à la VM.

Réseau en Pont (Bridged Network)

Le réseau en pont place la machine virtuelle sur le même réseau que la machine hôte. Cela signifie que la VM apparaît comme un appareil physique distinct sur le réseau, avec sa propre adresse IP qui

est souvent obtenue via DHCP. Les autres appareils sur le réseau peuvent communiquer directement avec la VM.

Réseau Interne (Internal Network)

Un réseau interne est un réseau virtuel privé aux VMs qui y sont connectées. Si vous choisissez cette option, les VMs peuvent communiquer entre elles comme si elles étaient sur le même réseau physique, mais elles ne peuvent pas accéder à l'hôte ou à Internet à moins que vous ne configuriez un service de routage ou de partage de connexion sur l'une des VMs.

Chaque type de connexion réseau a ses propres cas d'usage et implications en termes de performance, de sécurité et de connectivité. Il est important de choisir le type de réseau en fonction des besoins spécifiques de votre infrastructure et des objectifs de vos machines virtuelles.

ATELIER 3 : CONNEXION RÉSEAU DANS UNE VM

1- Configuration réseau dans VirtualBox :

Ouvrez les paramètres de votre VM et naviguez jusqu'à la section "Réseau".

Configurez deux adaptateurs réseau :

Adaptateur 1 : Attaché à NAT pour fournir une connexion Internet à la VM.

Adaptateur 2 : Attaché à "Pont" pour que la VM soit accessible depuis le réseau local.

2- Configuration de l'adresse IP :

Déterminez une adresse IP libre sur votre réseau local.

Configurez l'adaptateur réseau de la VM avec l'adresse IP choisie en suivant les étapes de la section précédente.

3- Test de connectivité :

Depuis la VM, utilisez ping pour vérifier la connectivité au routeur et à Internet.

Depuis l'hôte, utilisez ping pour vérifier la connectivité avec la VM.

LE STOCKAGE VIRTUEL : THÉORIE

Disques Durs Virtuels

Un disque dur virtuel (Virtual Hard Disk - VHD) est un fichier qui simule un disque dur physique. Il peut contenir tout ce qu'un vrai disque pourrait contenir, tels que des systèmes de fichiers, des fichiers, des dossiers et des systèmes d'exploitation. Ces fichiers de disque sont utilisés par les machines virtuelles (VM) pour stocker leurs données et pour simuler le démarrage et le fonctionnement d'un ordinateur physique.

Formats de Fichier de Disque Virtuel

Il existe plusieurs formats de fichier de disque dur virtuel, chacun ayant ses spécificités et compatibilités :

VDI (Virtual Disk Image) :

Utilisé par VirtualBox.

Il a l'avantage d'être optimisé pour le logiciel hôte, offrant une meilleure intégration avec les fonctionnalités de VirtualBox.

VHD (Virtual Hard Disk) :

Originellement développé par Connectix pour les machines virtuelles Microsoft.

Il est utilisé par Hyper-V et prend en charge les fonctionnalités spécifiques à Windows.

VMDK (Virtual Machine Disk) :

Développé par VMware pour ses produits de virtualisation.

Il est également pris en charge par d'autres hyperviseurs, y compris VirtualBox.

Ces formats ont des structures et des spécifications différentes, mais le concept est le même : ils agissent tous comme de vrais disques durs pour les VMs.

Stockage Alloué Statiquement vs Dynamiquement

Il y a deux méthodes principales pour allouer de l'espace à ces disques durs virtuels :

1- **Stockage Alloué Statiquement (Fixed Size) :**

Lors de la création d'un disque dur virtuel avec allocation statique, tout l'espace disque est alloué immédiatement.

Avantages :

- Meilleures performances, car l'hyperviseur n'a pas besoin d'allouer de l'espace disque supplémentaire pendant l'utilisation de la VM.
- Moins de fragmentation du disque sur l'hôte.

Inconvénients :

- Utilisation immédiate de l'espace disque, indépendamment du fait que la VM utilise ou non cet espace.
- Moins flexible en termes de gestion de l'espace disque.

2- **Stockage Alloué Dynamiquement (Dynamically Allocated) :**

Seul un petit espace est utilisé initialement, et le fichier de disque virtuel grandit au fur et à mesure que les données sont ajoutées à la VM.

Avantages :

- Économie d'espace disque initial puisque le fichier de disque virtuel ne grandit que lorsque cela est nécessaire.
- Plus flexible, permettant de créer plusieurs disques ou VMs sans consommer l'intégralité de l'espace de stockage disponible immédiatement.

Inconvénients :

- Peut entraîner une dégradation des performances car l'hyperviseur doit allouer de l'espace supplémentaire en temps réel.
- Risque de fragmentation de l'espace disque sur l'hôte.

Dans la pratique, le choix entre ces deux options dépendra des besoins spécifiques en termes de performances et de flexibilité de gestion de l'espace disque. Pour des environnements de production à haut débit, l'allocation statique est souvent préférée pour ses performances stables et prévisibles. En revanche, pour les environnements de développement ou de test où la flexibilité et l'économie d'espace sont primordiales, l'allocation dynamique est généralement le meilleur choix.

ATELIER 4 : GESTION DU STOCKAGE DANS VIRTUAL BOX

1- Accédez à la gestion des disques virtuels dans VirtualBox.

2- Créez un nouveau disque dur virtuel et attachez-le à la VM.

3- Partitionnez et formatez le nouveau disque dur dans la VM en utilisant fdisk ou gparted.

4- Montez le nouveau disque dans le système de fichiers de la VM et transférez des fichiers vers celui-ci.

ORDONNANCEURS : KUBERNETES

L'architecture de Kubernetes est conçue pour gérer et orchestrer des conteneurs à l'échelle de production.

Mais Kubernetes, souvent appelé K8S, va bien au-delà de la simple orchestration de conteneurs. Il s'agit d'une plate-forme open source conçue pour automatiser le déploiement, la mise à l'échelle et l'exploitation des conteneurs d'applications.

Là où Docker est à la traîne, Kubernetes excelle. Docker a révolutionné la conteneurisation, la rendant accessible et standardisée. Cependant, lorsqu'il s'agit de gérer un grand nombre de conteneurs sur différents serveurs, Docker peut ne pas être à la hauteur.

Kubernetes intervient ici, fournissant un environnement basé sur des clusters plus robuste pour gérer les applications conteneurisées à grande échelle. Il offre une haute disponibilité, un équilibrage de charge et un mécanisme d'autoréparation, garantissant que les applications sont toujours opérationnelles et distribuées efficacement.

Le principal problème résolu par Kubernetes est la complexité de la gestion de plusieurs conteneurs sur différents serveurs.

Il automatise la distribution et la planification des conteneurs sur un cluster, gère les exigences de mise à l'échelle et garantit un environnement cohérent entre le développement, les tests et la production.

L'interface CRI (Container Runtime Interface) de Kubernetes constitue un pas en avant significatif, permettant aux utilisateurs de connecter différents environnements d'exécution de conteneur sans recompiler Kubernetes.

Cette flexibilité signifie que les organisations peuvent choisir parmi une variété d'environnements d'exécution tels que Docker, containersd, CRI-O et autres, en fonction de leurs besoins spécifiques.

Clusters

Un cluster Kubernetes est un ensemble de machines, appelées nodes, qui exécutent des conteneurs. Le cluster est l'ensemble de l'infrastructure de calcul qui constitue l'environnement d'exécution de vos applications.

Nodes

Un node est une machine physique ou virtuelle sur laquelle Kubernetes peut planifier des conteneurs. Chaque node exécute le kubelet, qui est un agent supervisant que les conteneurs sont exécutés dans un Pod.

Il y a deux types de nodes :

Master Node : Il gère l'état du cluster, planifie les applications, et orchestre la communication entre les différents composants de Kubernetes. Le Master Node exécute plusieurs composants critiques :

- API Server : Le point d'entrée de l'API Kubernetes.
- Scheduler : Qui décide sur quel Node un Pod doit être exécuté.
- Controller Manager : Qui supervise les contrôleurs qui régulent l'état du cluster.
- etcd : Un magasin clé-valeur conservant toute la configuration du cluster et l'état des services.

Worker Node : Ils exécutent les applications sous la forme de conteneurs. Un Worker Node contient les services nécessaires pour communiquer avec le Master Node et gérer les ressources de conteneurisation :

- Kubelet : Qui communique avec le Master Node.
- Kube-Proxy : Qui gère le réseau des Pods et les communications avec l'extérieur.
- Container Runtime : L'environnement d'exécution des conteneurs, comme Docker ou rkt.

Pods

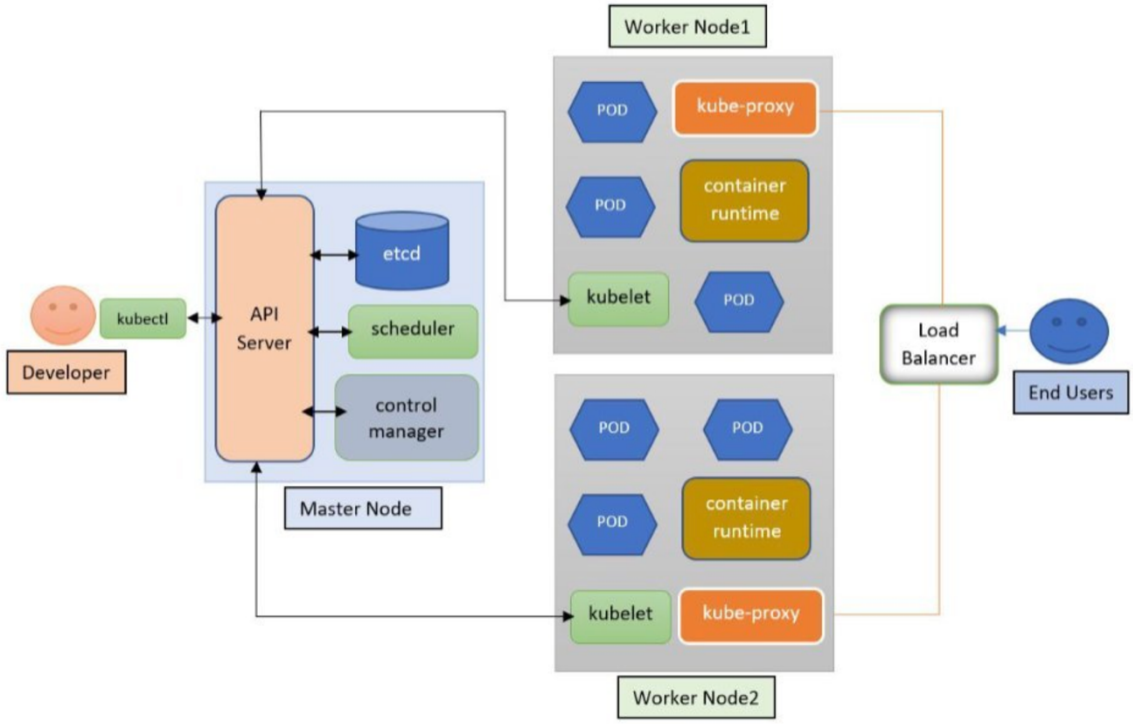
Un Pod est l'unité de base de déploiement dans Kubernetes. Il s'agit d'un groupe d'un ou plusieurs conteneurs qui partagent le même espace de stockage (Volumes), adresse IP et informations sur comment les exécuter. Les conteneurs d'un même Pod sont toujours co-localisés et co-planifiés sur le même node.

Schéma de l'architecture de Kubernetes :

Un cluster Kubernetes se compose d'un ensemble de machines de travail, appelées nœuds, qui exécutent des applications conteneurisées. Chaque cluster possède au moins un nœud de travail.

Les nœuds de travail hébergent les Pods, qui sont les composants de la charge de travail de l'application.

Le control plane (plan de contrôle) gère les nœuds de travail et les Pods dans le cluster.



Deployments

Un Deployment est une spécification déclarative pour les Pods. Il décrit l'état désiré pour les instances de votre application. Le Deployment utilise un ReplicaSet pour maintenir le nombre de Pods, en les redémarrant ou en les remplaçant si un node échoue ou si un Pod devient défectueux. C'est le moyen recommandé de gérer la création et l'échelonnage des Pods.

Services

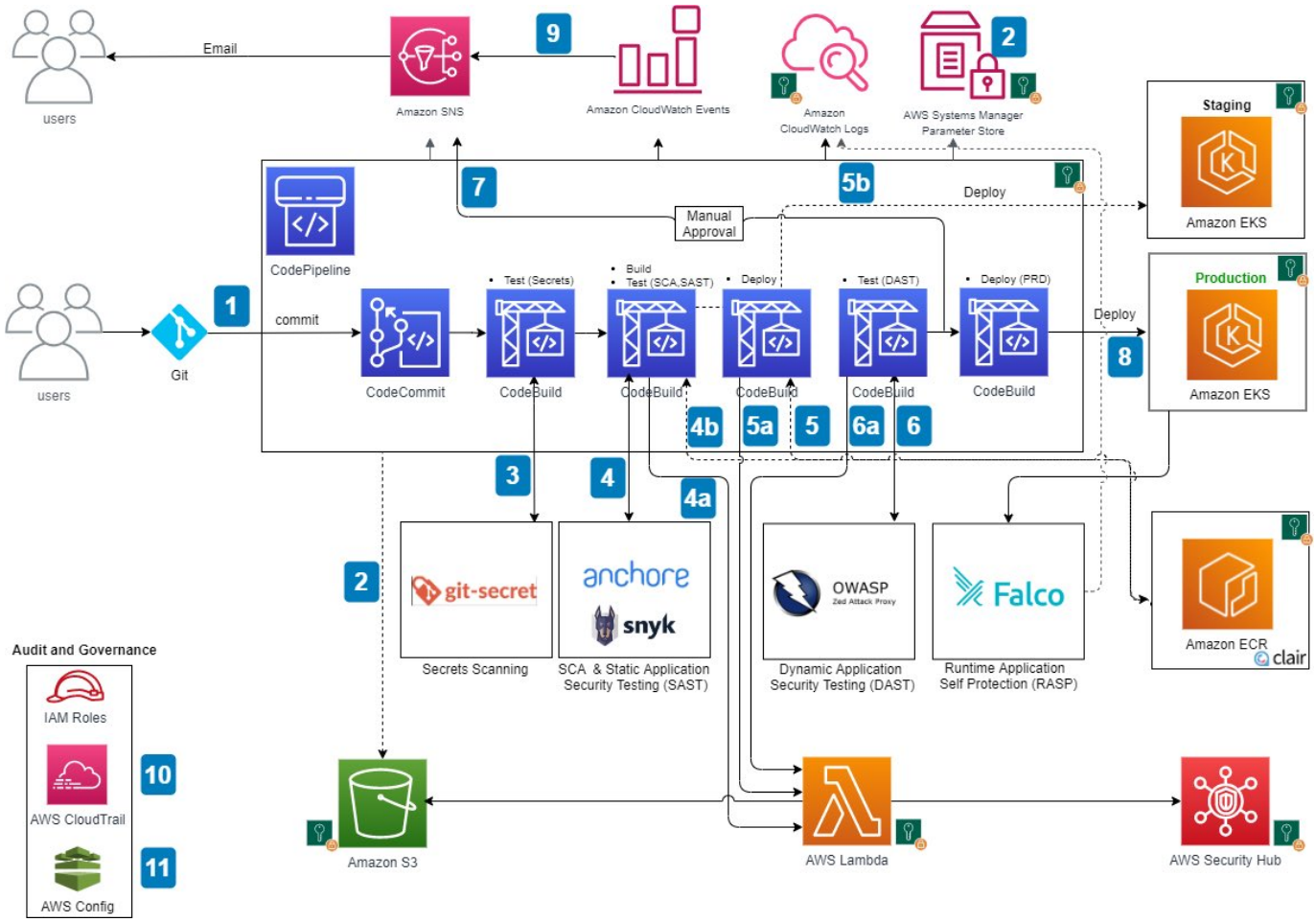
Un Service est une abstraction qui définit un ensemble logique de Pods et une politique d'accès à ceux-ci, généralement via une adresse IP fixe. Les Services permettent de fournir une adresse IP persistante pour un groupe de Pods, indépendamment des cycles de vie des Pods individuels. Cela signifie que même si les Pods sous-jacents changent, le point d'accès reste constant.

Autres Composants Importants

Ingress : Il gère l'accès externe aux services dans le cluster, généralement HTTP.

ConfigMaps et Secrets : Fournissent des configurations et des secrets aux Pods, permettant de séparer la configuration de l'image de conteneur pour promouvoir la portabilité.

Pipeline avec Kubernetes



ATELIER 5 : MINIKUBE

1- Installation de Minikube :

Minikube permet de créer un cluster Kubernetes local. Installez Minikube en suivant les instructions du site officiel.

2- Démarrage de Minikube :

Lancez Minikube avec la commande minikube start.

```
minikube start --driver=docker
```

3- Déploiement d'une application :

Utilisez kubectl, l'outil en ligne de commande pour Kubernetes, pour déployer une application.

Exemple de déploiement d'une application Nginx :

```
kubectl create deployment nginx --image=nginx
```

4- Exposition de l'application :

Exposez l'application Nginx à l'extérieur du cluster :

```
kubectl expose deployment nginx --type=NodePort --port=80
```

Utilisez minikube service nginx pour ouvrir la page d'accueil de Nginx dans un navigateur web ou cette commande :

```
minikube service nginx --url
```

5- Exploration de Kubernetes :

Familiarisez-vous avec l'interface de ligne de commande kubectl en exécutant diverses commandes pour explorer l'état du cluster, telles que kubectl get pods, kubectl get services, etc.

6- Nettoyage :

Après avoir terminé l'atelier, supprimez les ressources créées avec les commandes suivantes :

```
kubectl delete service nginx
kubectl delete deployment nginx
```