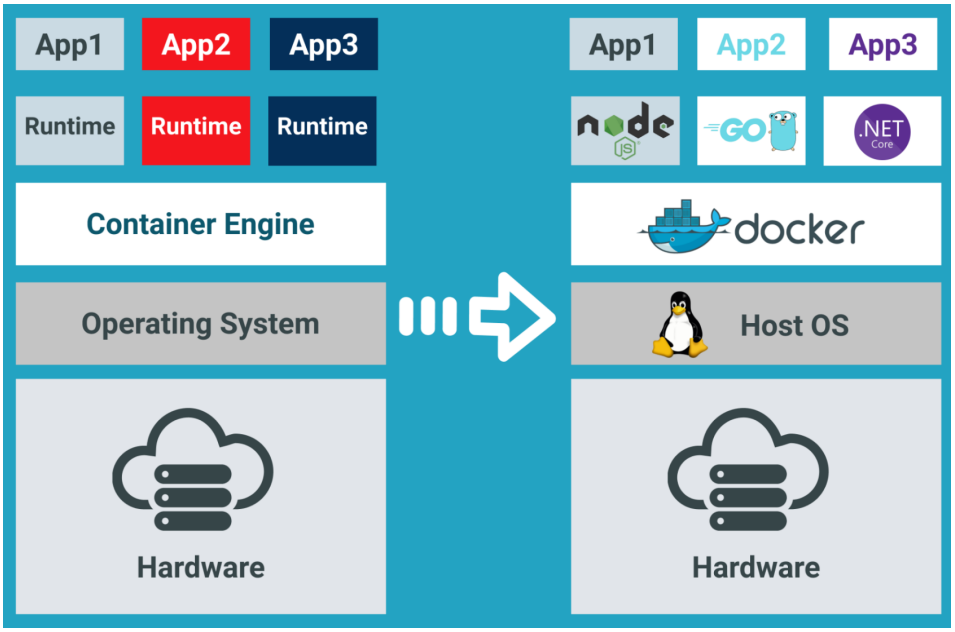
Présentation

Docker

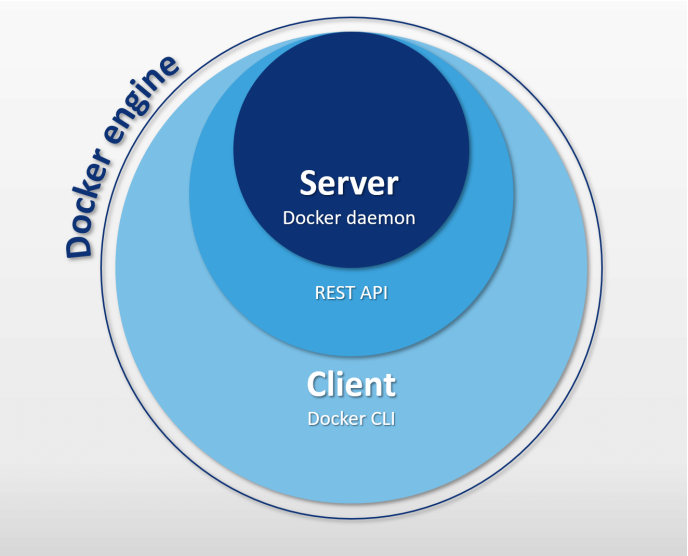
Qu'est-ce que Docker ?

Conteneurs : Docker utilise la virtualisation au niveau du système d'exploitation pour exécuter des applications dans des conteneurs. Un conteneur encapsule une application et ses dépendances dans un environnement exécutable, séparé du système hôte. Cela garantit que l'application fonctionne de manière uniforme dans n'importe quel environnement Docker.



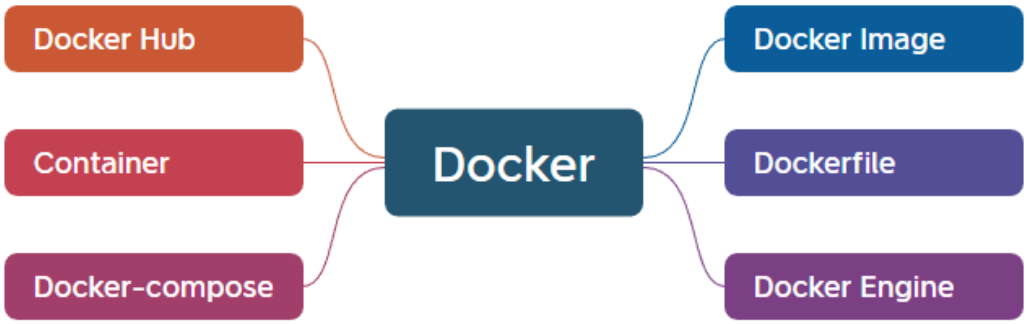
Images Docker : Les conteneurs sont créés à partir d'images Docker, qui sont des modèles immuables contenant le code de l'application, les bibliothèques, les outils, les dépendances, et toutes les autres nécessités pour exécuter une application. Les images sont stockées dans des registres Docker, comme Docker Hub, et peuvent être téléchargées et utilisées pour créer des conteneurs.

Docker Engine : Le moteur Docker est l'application client-serveur qui construit et exécute les conteneurs à l'aide de la technologie de conteneurisation du système d'exploitation. Il fournit l'environnement d'exécution pour les conteneurs et gère leur cycle de vie.



Dockerfile : Un fichier texte qui contient toutes les commandes qu'un utilisateur peut appeler sur la ligne de commande pour assembler une image Docker. Il simplifie le processus de création d'images en automatisant les étapes.

Environnement Docker



Utilité de Docker

Consistance et Isolation : Docker assure que votre application fonctionne de la même manière dans tous les environnements, depuis le développement jusqu'à la production, en isolant les applications dans des conteneurs. Cela élimine le problème du "fonctionne sur ma machine".

Rapidité et Légèreté : Les conteneurs Docker partagent le même noyau de système d'exploitation, mais s'exécutent comme des processus isolés. Cela les rend beaucoup plus légers et

plus rapides que les machines virtuelles traditionnelles.

Déploiement Facile : Docker permet un déploiement facile et rapide des applications. Avec Docker, les développeurs peuvent facilement empaqueter une application et ses dépendances dans une image Docker, qui peut ensuite être exécutée n'importe où.

Développement, Intégration et Déploiement Continus : Docker facilite les pratiques de CI/CD en permettant aux développeurs de créer et tester des applications dans des environnements conteneurisés, puis de déployer ces conteneurs dans des environnements de production.

Écosystème et Standardisation : Docker possède un vaste écosystème de technologies complémentaires pour le réseau, le stockage, la sécurité, etc. De plus, il a établi un standard dans l'industrie pour la conteneurisation, avec un large support et une communauté active.

Conclusion

Docker offre une solution puissante et flexible pour le développement et le déploiement d'applications. En utilisant la conteneurisation, Docker simplifie le processus de garantie que les applications fonctionneront dans n'importe quel environnement, tout en améliorant l'efficacité et la productivité des développeurs.

Que ce soit pour des applications simples ou des architectures de services complexes, Docker fournit les outils nécessaires pour construire, livrer et gérer des applications modernes avec moins de friction.

Un peu d'histoire

L'histoire de Docker est celle d'une innovation qui a révolutionné le monde du développement logiciel et de la conteneurisation. Docker, Inc., l'entreprise derrière cette technologie, a été fondée par Solomon Hykes, Sebastien Pahl, et Kamel Founadi.

Le voyage de Docker a commencé en mars 2013, lorsqu'il a été lancé comme un projet open source lors de la conférence PyCon.

Les Débuts

2013 : Docker est introduit au public. À l'origine, Docker utilisait la technologie LXC (Linux Containers) pour la virtualisation au niveau du système d'exploitation, permettant aux conteneurs de s'exécuter isolément sur un même hôte Linux. L'idée était de permettre aux développeurs d'empaqueter une application et ses dépendances dans un conteneur virtuel qui pourrait être exécuté sur n'importe quel système Linux.

Croissance Rapide et Adoption

2014-2015 : Docker gagne rapidement en popularité dans la communauté du développement logiciel. Son approche innovante de la conteneurisation simplifie le déploiement d'applications, ce qui attire l'attention des développeurs et des entreprises du monde entier. Docker, Inc. reçoit d'importants investissements, ce qui lui permet d'étendre son équipe et d'accélérer le développement de nouvelles fonctionnalités.

Évolution Technique

2014 : Docker introduit sa propre interface pour la gestion des conteneurs, appelée libcontainer, remplaçant LXC pour offrir une meilleure portabilité et intégration. Cette période voit également le lancement de Docker Hub, un service de registre d'images Docker qui facilite le partage et la distribution d'images conteneurisées.

Expansion de l'Écosystème

2015 et au-delà : L'écosystème Docker s'étend avec le développement d'outils complémentaires tels que Docker Compose pour la définition et l'exécution d'applications multi-conteneurs, Docker Swarm pour l'orchestration de conteneurs, et l'introduction de Docker for Windows et Docker for Mac, améliorant l'expérience des développeurs sur ces plateformes.

Docker et l'Orchestration de Conteneurs

2017 : Docker annonce le support natif de Kubernetes, l'outil d'orchestration de conteneurs open source de Google, dans Docker Enterprise. Cette décision reflète la popularité croissante de

Kubernetes dans la gestion de déploiements conteneurisés à grande échelle et marque un moment important dans l'histoire de Docker, soulignant sa volonté de s'adapter aux besoins de l'industrie.

Changements Organisationnels

2019-2020 : Docker, Inc. restructure ses activités pour se concentrer davantage sur les développeurs et les petites équipes, vendant sa division entreprise à Mirantis, une entreprise de services cloud. Cette période marque également un recentrage sur le développement de Docker Desktop et Docker Hub, ainsi que sur l'amélioration de l'expérience de développement logiciel avec Docker.

L'Impact de Docker

Docker a non seulement simplifié le développement et le déploiement d'applications mais a également joué un rôle clé dans la popularisation de la conteneurisation et des architectures microservices. En rendant les conteneurs accessibles et faciles à utiliser, Docker a permis aux équipes de développement de créer des applications plus modulaires, évolutives et faciles à gérer.

En résumé, l'histoire de Docker est marquée par une croissance rapide, une innovation continue et une capacité à façonner et à répondre aux tendances de l'industrie du logiciel. Malgré les défis et les changements dans son parcours, Docker reste une pierre angulaire dans l'écosystème du développement logiciel moderne, facilitant la vie des développeurs et influençant la manière dont les applications sont construites, déployées et gérées à l'échelle mondiale.

Caractéristiques

Conteneurisation et Isolation

- **Conteneurs légers** : Docker utilise la technologie des conteneurs pour isoler les applications et leurs environnements d'exécution, réduisant ainsi les conflits entre les applications et maximisant la compatibilité.
- **Partage du système d'exploitation** : Contrairement aux machines virtuelles, les conteneurs Docker partagent le même noyau système d'exploitation, mais restent isolés les uns des autres, ce qui les rend beaucoup plus légers et plus rapides.

Développement et Déploiement Simplifiés

- **Développement cohérent** : Docker assure que les applications fonctionnent de la même manière dans tous les environnements, depuis le développement jusqu'à la production, en éliminant le problème du "ça marchait sur ma machine".
- **Intégration et déploiement continus** : Docker s'intègre facilement avec les pipelines CI/CD, facilitant l'intégration et le déploiement continus des applications.

Écosystème et Portabilité

- **Docker Hub** : La plateforme offre un registre public appelé Docker Hub, où les utilisateurs peuvent télécharger et partager des images de conteneurs, rendant facile l'accès à des logiciels pré-packagés et leur distribution.
- **Portabilité** : Les conteneurs Docker peuvent être exécutés sur n'importe quel système d'exploitation supportant Docker, que ce soit sur des ordinateurs personnels, des serveurs physiques, des instances cloud, ou même des plateformes d'orchestration comme Kubernetes.

Gestion des Images et des Conteneurs

- **Images Docker** : Les applications et leurs dépendances sont empaquetées dans des images Docker, qui sont des modèles immuables utilisés pour créer des conteneurs.
- **Gestion facile des conteneurs** : Docker fournit des commandes simples pour gérer le cycle de vie des conteneurs, y compris leur création, exécution, arrêt, et suppression.

Sécurité

- **Isolation des applications** : L'isolation fournie par les conteneurs aide à limiter les risques de sécurité entre les applications.
- **Gestion des secrets** : Docker offre des mécanismes pour gérer de manière sécurisée les secrets et les configurations sensibles nécessaires aux applications.

Communauté et Support

- **Communauté active** : Docker bénéficie d'une large et active communauté d'utilisateurs et de développeurs qui contribuent à son développement, offrent du support et partagent des meilleures pratiques.
- **Documentation et ressources d'apprentissage** : Docker fournit une documentation complète, des tutoriels et des guides pour aider les nouveaux utilisateurs à démarrer et à exploiter pleinement la plateforme.

En résumé, Docker a transformé le paysage du développement logiciel en offrant une solution efficace pour la conteneurisation des applications, garantissant leur cohérence à travers différents environnements de développement, de test et de production.

Sa facilité d'utilisation, sa portabilité, et son vaste écosystème en font un outil indispensable pour les développeurs, les administrateurs système et les équipes DevOps cherchant à améliorer l'efficacité et la fiabilité de leurs processus de développement et de déploiement d'applications.

YAML

Qu'est-ce que YAML ?

YAML, qui signifie "YAML Ain't Markup Language" (une récursivité pour "YAML n'est pas un langage de balisage"), est un format de sérialisation de données lisible par l'homme, souvent utilisé pour écrire des fichiers de configuration ou pour échanger des données entre langages de programmation.

Sa simplicité et sa lisibilité ont fait de YAML un choix populaire pour de nombreux outils de développement, déploiement, et automatisation, y compris Docker Compose, les configurations de CI/CD comme GitHub Actions ou GitLab CI, et les systèmes de gestion de configuration comme Ansible.

Caractéristiques

Lisibilité : YAML est conçu pour être facile à lire et à comprendre par les humains, utilisant une indentation pour représenter la structure des données (semblable à Python).

Polyvalence : Il peut représenter les structures de données communes, telles que les scalaires (p. ex., chaînes, nombres), les listes (arrays), et les dictionnaires (objets), ce qui le rend très flexible pour différents cas d'usage.

Compatibilité : YAML est interopérable avec différents langages de programmation, permettant de sérialiser et désérialiser facilement des structures de données complexes.

Utilité de YAML

Fichiers de Configuration : Sa lisibilité et sa simplicité font de YAML un choix idéal pour les fichiers de configuration, où il est important que les configurations soient compréhensibles et modifiables facilement.

Définition d'Infrastructures et de Déploiements : Des outils comme Docker Compose et Kubernetes utilisent YAML pour permettre aux développeurs et aux opérateurs de définir des infrastructures de conteneurs, des réseaux, des volumes, et des politiques de déploiement de manière déclarative.

Automatisation et Orchestration : YAML est largement utilisé dans des systèmes d'automatisation et d'orchestration comme Ansible pour décrire les tâches d'automatisation, les configurations système, et les déploiements d'applications.

CI/CD : Les pipelines d'intégration et de déploiement continus sont souvent configurés via des fichiers YAML, permettant de définir les étapes du pipeline, les environnements, et les scripts

d'exécution de manière claire et structurée.

Syntaxe de base de YAML

Voici quelques éléments clés de la syntaxe YAML :

Indentation : YAML utilise l'indentation pour représenter la hiérarchie des données. L'indentation doit être faite avec des espaces, pas avec des tabulations.

Listes : Les éléments d'une liste sont précédés d'un tiret (`-`).

Dictionnaires (Maps) : Les dictionnaires sont des ensembles de paires clé-valeur, où chaque paire est séparée par deux points (`:`).

Commentaires : Les commentaires commencent par un dièse (`#`) et s'étendent jusqu'à la fin de la ligne.

Exemple d'un docker-compose.yml

```
version: '3.8'  # Spécifie la version de la syntaxe Docker Compose utilisée

services:  # Définit les services, c'est-à-dire les conteneurs à exécuter
  web:  # Nom du service web
    image: nginx:latest  # Utilise l'image Docker officielle de Nginx
    ports:
      - "80:80"  # Redirige le port 80 du conteneur vers le port 80 de l'hôte
    volumes:
      - ./html:/usr/share/nginx/html  # Montage d'un volume pour personnaliser le contenu servi par Nginx
    depends_on:
      - db  # Indique que le service web dépend de la base de données et doit attendre son démarrage

  db:  # Nom du service de base de données
    image: postgres:13  # Utilise l'image Docker officielle de PostgreSQL version 13
    volumes:
      - db_data:/var/lib/postgresql/data  # Utilise un volume nommé pour la persistance des données de la base de données
    environment:  # Définit les variables d'environnement pour la configuration de la base de données
      POSTGRES_DB: exampledb  # Nom de la base de données à créer
      POSTGRES_USER: exampleuser  # Nom de l'utilisateur de la base de données
      POSTGRES_PASSWORD: examplepass  # Mot de passe de l'utilisateur de la base de données

volumes:  # Déclare les volumes utilisés par les services
  db_data:  # Nom du volume pour la persistance des données PostgreSQL
```

Dans cet exemple, deux services sont définis :

1. **Service web (`web`)** :
   - Utilise l'image `nginx:latest` pour créer un conteneur Nginx servant de serveur web.
   - Les requêtes sur le port 80 de l'hôte sont redirigées vers le port 80 du conteneur, permettant d'accéder au serveur web depuis l'extérieur du conteneur.
   - Un volume est monté pour personnaliser le contenu servi par Nginx, permettant d'ajouter ou de modifier les fichiers HTML dans le dossier `./html` de l'hôte.
2. **Service de base de données (`db`)** :
   - Utilise l'image `postgres:13` pour créer un conteneur PostgreSQL servant de base de données.
   - Les données de la base de données sont stockées dans un volume nommé `db_data`, assurant leur persistance indépendamment du cycle de vie du conteneur.
   - Les variables d'environnement sont utilisées pour configurer la base de données, y compris le nom de la base de données, l'utilisateur et le mot de passe.

Ce fichier `docker-compose.yml` montre comment utiliser Docker Compose pour configurer et lier ensemble une application web simple et un service de base de données, illustrant l'utilité des volumes, des ports, des dépendances entre services, et des variables d'environnement dans la configuration de services Docker.

Dockerfile

Qu'est-ce qu'un Dockerfile ?

Un `Dockerfile` est un fichier texte qui contient toutes les commandes qu'un utilisateur peut appeler sur la ligne de commande pour assembler une image Docker. Il sert de script pour automatiser le processus de création d'images Docker, en définissant un ensemble d'étapes successives qui doivent être exécutées pour construire l'image finale.

Structure d'un Dockerfile

Un Dockerfile typique inclut une série de directives, chacune spécifiant une partie de l'image Docker et comment elle doit être construite.

Voici quelques-unes des directives les plus courantes :

- `FROM` : Définit l'image de base à utiliser pour l'image Docker. C'est souvent une distribution Linux légère comme Alpine ou Debian, ou une image préexistante contenant déjà l'environnement nécessaire à votre application.
- `RUN` : Exécute une commande dans le conteneur. Utilisé pour installer des logiciels, créer des fichiers, etc.
- `COPY` / `ADD` : Copie des fichiers et des répertoires de votre système de fichiers local dans le conteneur.
- `CMD` : Fournit la commande par défaut à exécuter lorsque le conteneur démarre.
- `ENTRYPOINT` : Configure une commande qui ne sera pas remplacée au démarrage du conteneur. `CMD` peut être utilisé pour fournir des arguments par défaut à `ENTRYPOINT`.
- `ENV` : Définit des variables d'environnement.
- `EXPOSE` : Indique les ports sur lesquels un conteneur écoute pour les connexions.
- `VOLUME` : Crée un point de montage pour accéder à et stocker les données persistantes.
- `WORKDIR` : Définit le répertoire de travail pour les instructions `CMD`, `RUN`, `ENTRYPOINT`, `COPY` et `ADD`.

Exemple :

```
# Utilise une image de base officielle de Python 3.8
FROM python:3.8-slim

# Définit le répertoire de travail dans le conteneur
WORKDIR /app

# Copie les fichiers requirements.txt dans le conteneur pour installer les dépendances
```

```
COPY requirements.txt ./

# Installe les dépendances Python spécifiées dans requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copie le reste du code source de l'application dans le conteneur
COPY . .

# Expose le port 5000 sur lequel l'application va s'exécuter
EXPOSE 5000

# Commande exécutée au démarrage du conteneur pour lancer l'application
CMD ["python", "./app.py"]
```

Dans cet exemple, le Dockerfile crée une image pour une application Python simple :

- Il commence par utiliser l'image Docker officielle de Python 3.8 comme base.
- Il définit `/app` comme répertoire de travail dans le conteneur.
- Il copie `requirements.txt` dans le conteneur et installe les dépendances Python nécessaires.
- Il copie ensuite le reste du code source de l'application dans le conteneur.
- Il expose le port 5000, sur lequel l'application va écouter.
- Enfin, il définit la commande par défaut pour exécuter l'application avec Python.

Ce Dockerfile illustre comment préparer un environnement pour une application Python, en installant des dépendances et en configurant l'application pour qu'elle soit prête à être exécutée dans un conteneur Docker.

Image Docker

Qu'est-ce qu'une Image Docker ?

Les images Docker sont des modèles immuables utilisés pour créer des conteneurs Docker. Elles constituent la base de la conteneurisation avec Docker, permettant de paquetiser le code, les outils, les bibliothèques, les dépendances, et tous les fichiers nécessaires à l'exécution d'une application dans un environnement isolé.

Les images Docker sont essentielles pour assurer la portabilité, la reproductibilité et la scalabilité des applications dans divers environnements de déploiement.

Caractéristiques des Images Docker

- **Immutabilité** : Une fois une image créée, elle ne change pas. Les modifications sont appliquées en créant une nouvelle image, ce qui favorise la consistance et la reproductibilité des environnements d'exécution.
- **Architecture en couches** : Les images Docker sont construites en utilisant un système de fichiers en couches. Chaque instruction dans un Dockerfile crée une nouvelle couche dans l'image. Les couches sont réutilisées entre les images, ce qui optimise l'utilisation de l'espace disque et le temps de téléchargement des images.
- **Stockage et partage** : Les images Docker peuvent être stockées et partagées à travers des registres d'images publics ou privés, comme Docker Hub, facilitant le partage et la distribution d'applications et de services conteneurisés.

Création et Utilisation des Images Docker

Création d'une Image

La création d'une image Docker commence généralement par l'écriture d'un Dockerfile, qui définit les étapes nécessaires pour assembler l'image, incluant la base à utiliser, les fichiers à copier, les commandes à exécuter, etc. Une fois le Dockerfile prêt, une image peut être créée en exécutant la commande :

```
docker build -t monimage:tag .
```

Cette commande construit une nouvelle image Docker à partir du Dockerfile dans le répertoire courant, en lui attribuant un nom (`monimage`) et un tag (`tag`).

Utilisation d'une Image

Pour exécuter un conteneur basé sur une image Docker, on utilise la commande :

```
docker run -d --name monconteneur monimage:tag
```

Cela crée et démarre un nouveau conteneur nommé `monconteneur` à partir de l'image `monimage:tag`. Le conteneur s'exécute en arrière-plan (`-d` pour "detached").

Gestion des Images Docker

Les images Docker peuvent être listées, modifiées, supprimées, et partagées à travers des commandes Docker. Par exemple :

- **Lister les images** : `docker images` ou `docker image ls`
- **Supprimer une image** : `docker rmi monimage:tag`
- **Télécharger une image** : `docker pull monimage:tag`
- **Envoyer une image** dans un registre : `docker push monimage:tag`

Exemple création d'une image docker :

Dockerfile

```
# Définir l'image de base. Ici, on utilise l'image officielle Python 3.8 en version "slim" pour une image finale plus légère.
FROM python:3.8-slim

# Définir le répertoire de travail dans le conteneur. Toutes les commandes qui suivent seront exécutées dans ce répertoire.
WORKDIR /app

# Copier le fichier requirements.txt dans le répertoire de travail (/app) du conteneur. Ce fichier liste les dépendances de l'application.
COPY requirements.txt .

# Installer les dépendances Python listées dans requirements.txt. L'option --no-cache-dir est utilisée pour minimiser la taille de l'image.
RUN pip install --no-cache-dir -r requirements.txt

# Copier les autres fichiers de l'application du répertoire courant sur l'hôte dans le répertoire de travail du conteneur.
COPY . .

# Exposer le port sur lequel l'application va s'exécuter. Cela ne publie pas le port, mais sert de documentation entre celui qui déploie l'image et l'image elle-même.
EXPOSE 5000
```

```
# Définir la commande par défaut pour exécuter l'application. Ici, on lance l'application Flask en utilisant le serveur de développement de Flask.
CMD ["flask", "run", "--host=0.0.0.0"]
```

Pour cet exemple, supposons que vous avez une application Flask simple. Le fichier `requirements.txt` pourrait ressembler à ceci :

```
Flask==1.1.2
```

Et une application Flask de base, `app.py`, pourrait ressembler à :

Après avoir créé votre `Dockerfile` et placé `requirements.txt` et `app.py` dans le même dossier, vous pouvez construire l'image Docker avec la commande suivante (n'oubliez pas de remplacer `<tag>` par le nom et le tag de votre choix) :

```
docker build -t monapplicationflask:<tag> .
```

Cette commande construira une image Docker basée sur le `Dockerfile` du répertoire courant, en téléchargeant l'image Python, en installant les dépendances, et en copiant les fichiers de l'application dans l'image.

Pour exécuter un conteneur basé sur votre image nouvellement créée, utilisez :

```
docker run -p 5000:5000 monapplicationflask:<tag>
```

Cela démarrera un conteneur de votre image `monapplicationflask`, mappant le port 5000 du conteneur au port 5000 de votre hôte, permettant ainsi d'accéder à votre application Flask en naviguant vers `http://localhost:5000` dans votre navigateur.

Conclusion

Les images Docker sont au cœur de la plateforme Docker, permettant de créer des conteneurs qui exécutent des applications de manière isolée et reproductible.

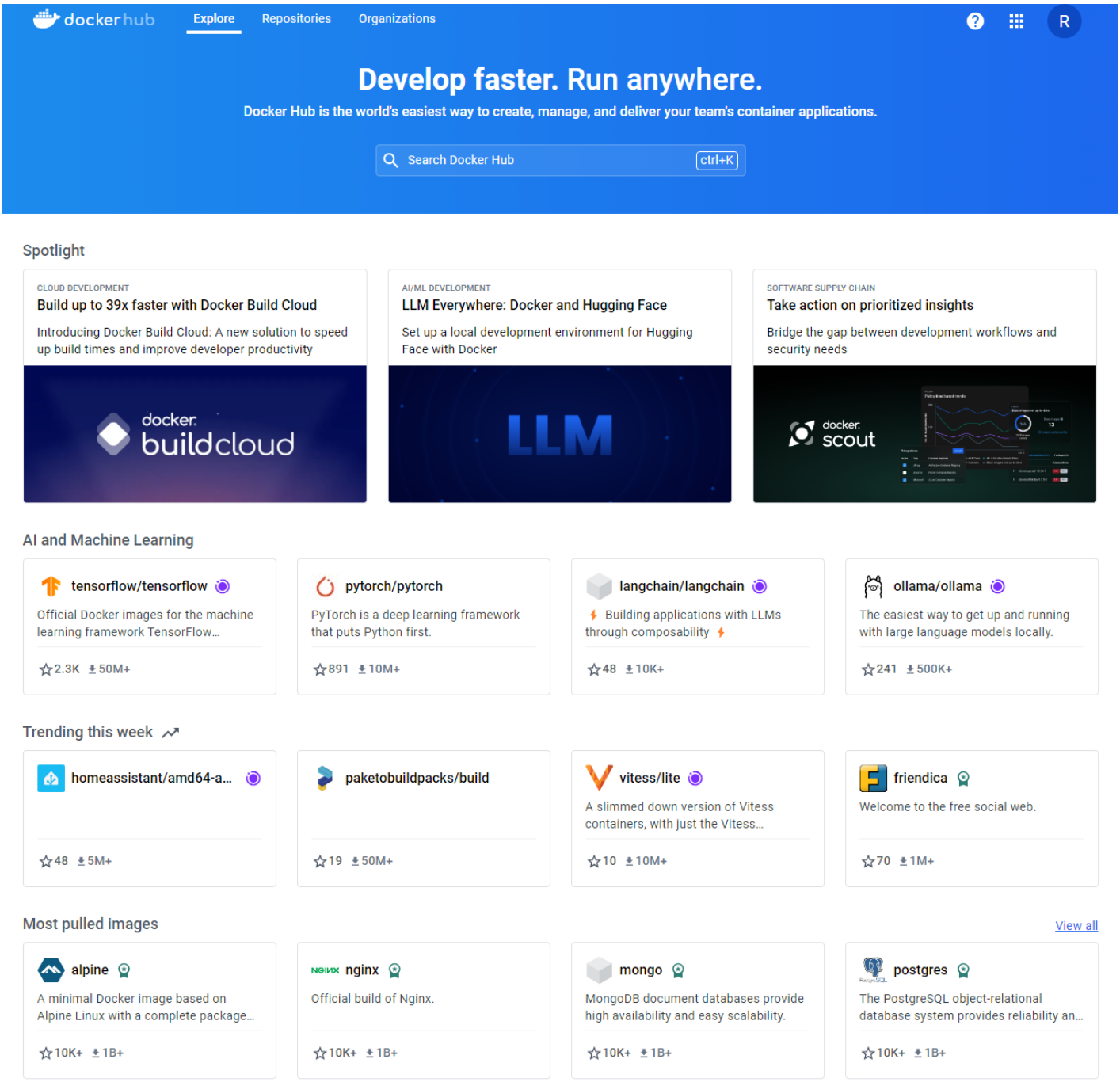
Grâce à leur architecture en couches, leur immutabilité, et la facilité avec laquelle elles peuvent être partagées et stockées, les images Docker simplifient le déploiement et la gestion des applications dans des environnements de développement, de test, et de production.

Docker Hub

Qu'est-ce que Docker Hub ?

Docker Hub est une plateforme de service de registre gérée qui permet aux développeurs et aux équipes DevOps de partager et de distribuer des conteneurs Docker.

C'est un service centralisé pour trouver et partager des images Docker, offrant à la fois des répertoires publics et privés pour gérer et stocker des images Docker. Voici un aperçu plus détaillé du rôle et des fonctionnalités de Docker Hub.



Rôle principal de Docker Hub

- **Répertoire d'Images Docker** : Docker Hub sert de répertoire central où les utilisateurs peuvent pousser, tirer, et gérer des images Docker. Il contient une vaste collection d'images Docker provenant de développeurs individuels, d'équipes open-source, et d'entreprises. Cela inclut des images officielles pour des logiciels populaires comme MySQL, Redis, et Ubuntu, ainsi que des images communautaires fournies par des utilisateurs du monde entier.

- **Partage et Collaboration** : Docker Hub facilite le partage d'images Docker entre les membres d'une équipe ou avec la communauté plus large. Les utilisateurs peuvent télécharger (push) leurs images sur Docker Hub pour les rendre accessibles aux autres, ou télécharger (pull) des images de Docker Hub pour les utiliser dans leurs propres projets et déploiements.

- **Intégration avec Docker** : Docker Hub est étroitement intégré à l'écosystème Docker, permettant aux utilisateurs de se connecter facilement à leurs comptes Docker Hub depuis la ligne de commande Docker pour pousser et tirer des images.

- **Gestion des Versions et Automatisation** : Docker Hub supporte le versionnage des images avec des tags, permettant aux utilisateurs de gérer différentes versions d'une même image et d'automatiser les déploiements en utilisant des tags spécifiques.

- **Webhooks et Automatisation CI/CD** : Docker Hub offre la possibilité de configurer des webhooks, qui peuvent déclencher des actions automatiques (comme des déploiements ou des mises à jour) dans des systèmes externes chaque fois qu'une nouvelle image est poussée dans un répertoire.

Fonctionnalités clés de Docker Hub

- **Images Officielles et Certifiées** : Fournit des images de haute qualité, sécurisées et maintenues par des organisations et des contributeurs fiables.

- **Répertoires Privés** : Permet aux utilisateurs et aux organisations de stocker et de gérer des images Docker privées, accessibles uniquement par des utilisateurs ou des équipes autorisées.

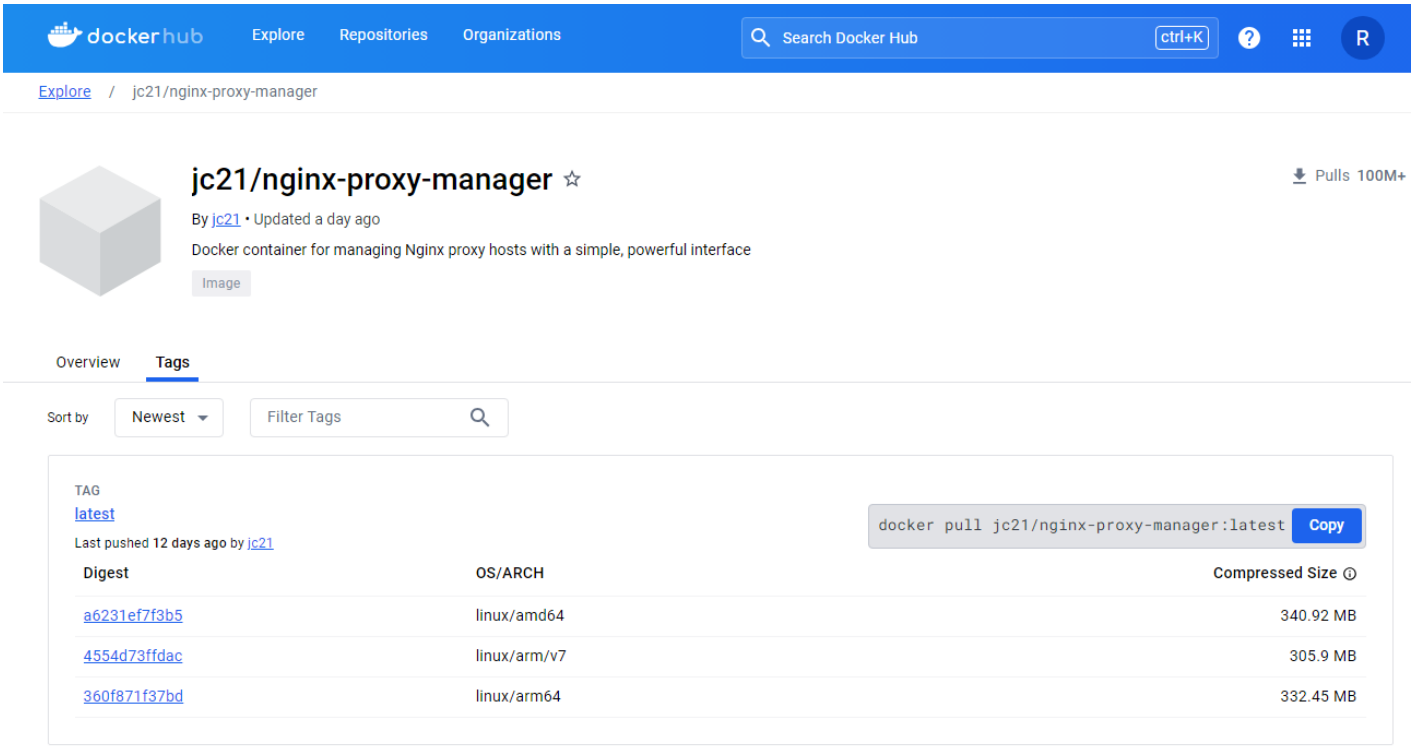
- **Automatisation des Builds** : Docker Hub peut automatiquement construire des images Docker à partir de code source stocké dans des dépôts GitHub ou Bitbucket chaque fois que des modifications sont apportées, simplifiant le processus d'intégration et de déploiement continus (CI/CD).

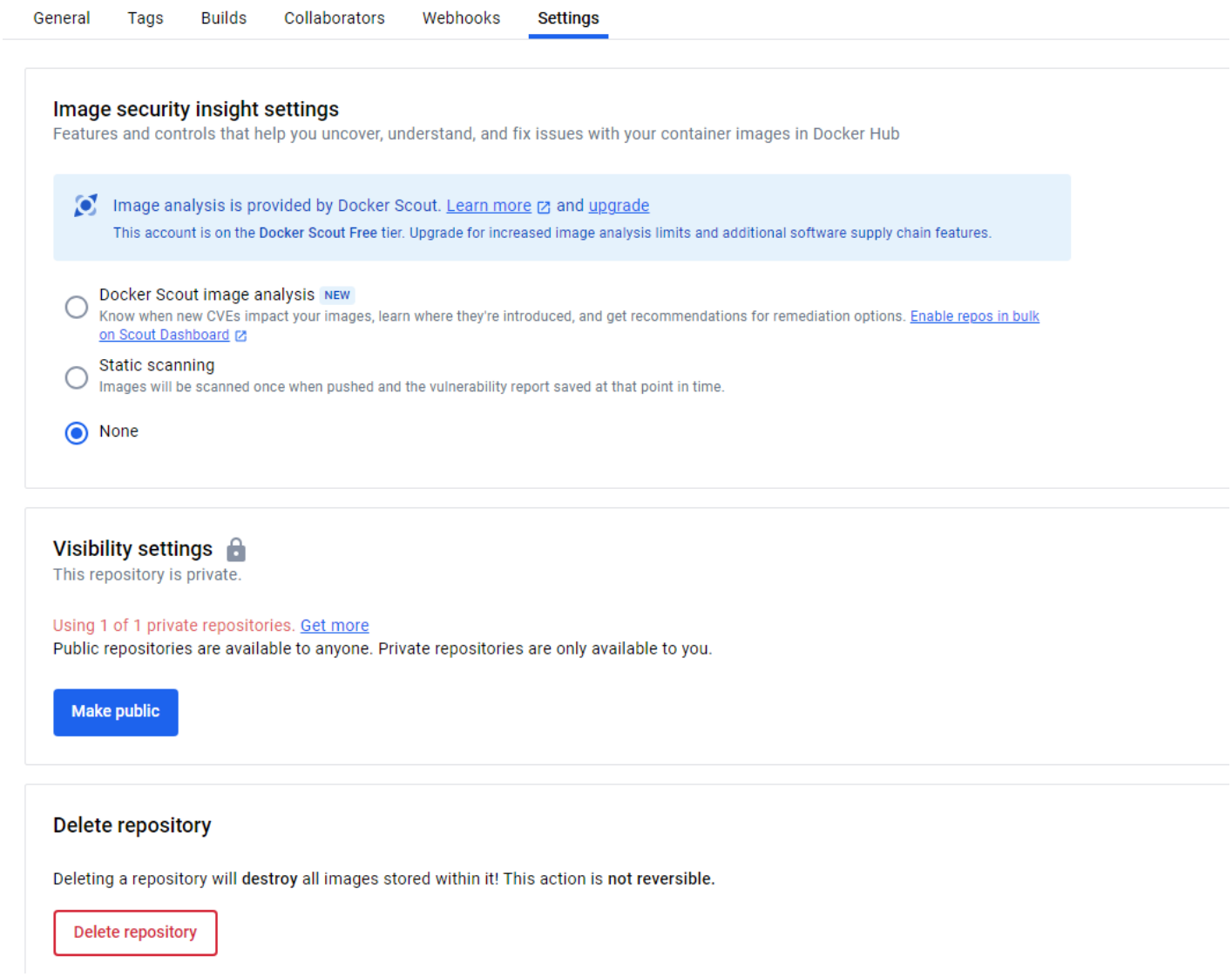
- **Gestion des Utilisateurs et des Groupes** : Les organisations peuvent gérer l'accès aux répertoires privés en configurant des groupes d'utilisateurs et des permissions spécifiques.

En images

Sur Docker Hub, vous avez la flexibilité de contrôler l'accès à vos images Docker en choisissant de rendre vos dépôts publics ou privés.

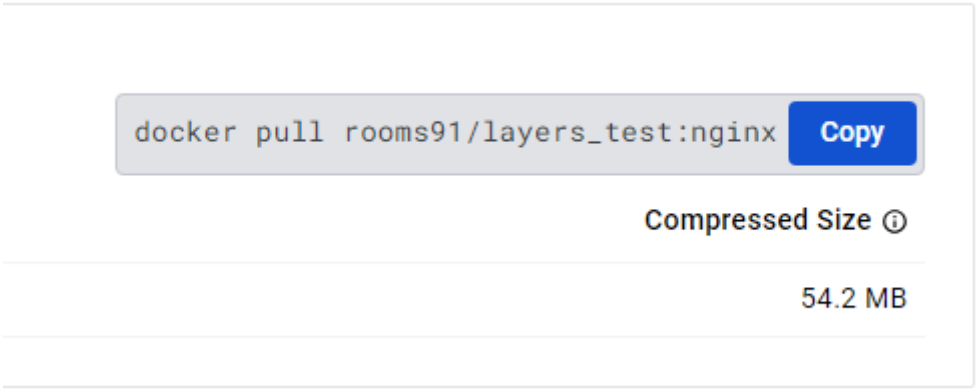
Cette fonctionnalité vous permet de partager librement vos images avec la communauté ou de les réserver exclusivement pour votre usage personnel ou celui de votre équipe.





Dans cet exemple, je vais vous montrer comment télécharger une image depuis un dépôt privé en exécutant une commande `docker pull`, tout en m'authentifiant avec mes identifiants.

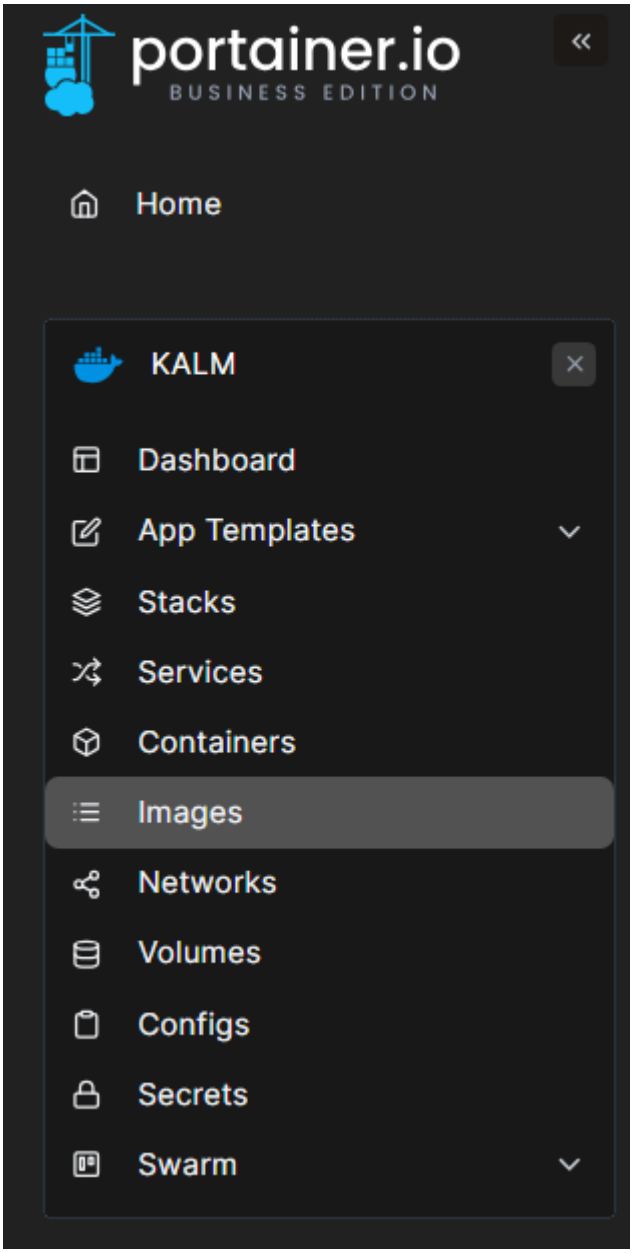
```
root@EVA-00:~# docker login -u rooms91
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
root@EVA-00:~# docker pull rooms91/layers_test:nginx
nginx: Pulling from rooms91/layers_test
e9995326b091: Pull complete
71689475aec2: Pull complete
f88a23025338: Pull complete
0df440342e26: Pull complete
eef26ceb3309: Pull complete
8e3ed6a9e43a: Pull complete
Digest: sha256:06aa2038b42f1502b59b3a862b1f5980d3478063028d8e968f0810b9b0502380
Status: Downloaded newer image for rooms91/layers_test:nginx
docker.io/rooms91/layers_test:nginx
```

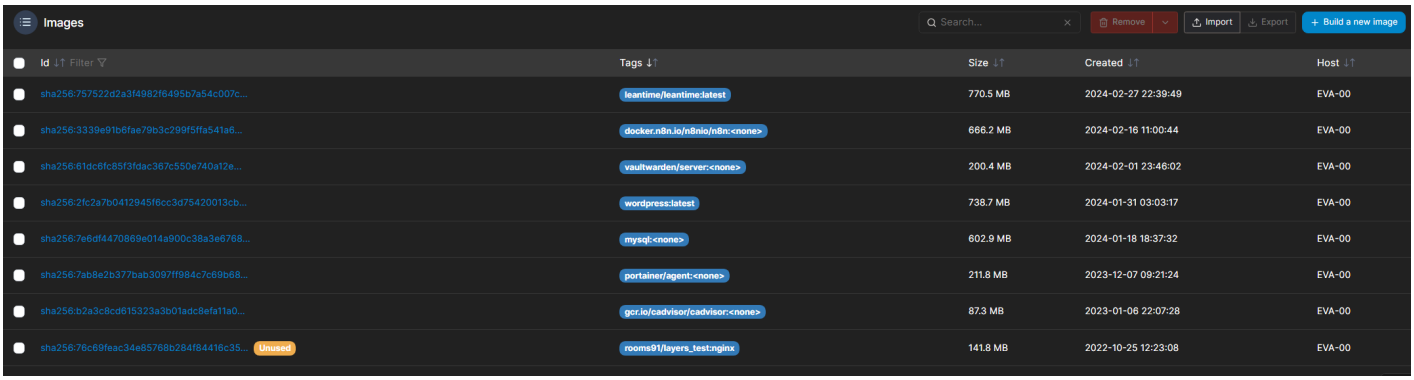


Pour afficher la liste des images Docker téléchargées sur votre système, utilisez la commande `docker images` ou son alias `docker image ls`.

```
root@EVA-00:~# docker image ls
REPOSITORY                 TAG       IMAGE ID       CREATED          SIZE
leantime/leantime          latest    757522d2a3f4   12 days ago      771MB
docker.n8n.io/n8nio/n8n    <none>    3339e91b6fae   3 weeks ago      666MB
vaultwarden/server         <none>    61dc6fc85f3f   5 weeks ago      200MB
wordpress                  latest    2fc2a7b04129   5 weeks ago      739MB
mysql                      <none>    7e6df4470869   7 weeks ago      603MB
portainer/agent            <none>    7ab8e2b377ba   3 months ago     212MB
gcr.io/cadvisor/cadvisor   <none>    b2a3c8cd6153   14 months ago    87.3MB
rooms91/layers_test        nginx     76c69feac34e   16 months ago    142MB
root@EVA-00:~#
```

Vous pouvez également visualiser les images Docker téléchargées sur votre système via une interface graphique en utilisant Portainer. Après avoir configuré Portainer, accédez à la section "Images" pour voir la liste des images.





> Site web : Docker Hub

Conclusion

Docker Hub joue un rôle central dans l'écosystème Docker en servant de hub pour la distribution, le partage, et la gestion des images Docker.

Que vous soyez un développeur travaillant sur un projet open-source, une petite équipe développant une application, ou une grande entreprise déployant des services à l'échelle, Docker Hub offre les outils et les services nécessaires pour faciliter le travail avec des conteneurs Docker.

Docker Compose

Qu'est-ce que Docker Compose ?

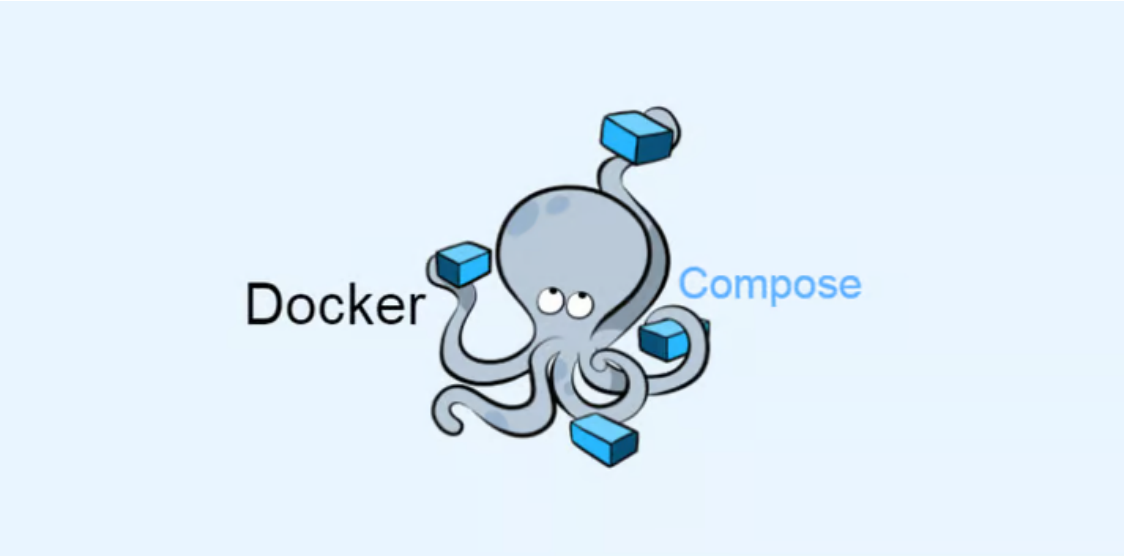
Docker Compose est un outil pour définir et exécuter des applications multi-conteneurs Docker. Il permet aux développeurs de déclarer, dans un fichier **YAML**, l'ensemble des services, réseaux et volumes nécessaires à leur application, simplifiant ainsi le processus de configuration et d'exécution de l'application dans un environnement Docker. Docker Compose se concentre sur l'automatisation du déploiement d'applications composées de multiples conteneurs qui doivent travailler ensemble de manière cohérente.

Fichier **`docker-compose.yml`** : Au cœur de Docker Compose se trouve le fichier `docker-compose.yml`, qui décrit les services qui composent l'application, y compris leur configuration, les réseaux personnalisés, et les volumes. Ce fichier YAML permet une définition déclarative de toute l'infrastructure nécessaire à l'application, rendant le déploiement reproductible et facile à comprendre.

Services : Dans le contexte de Docker Compose, un service est une application ou un processus exécuté dans un conteneur. Un fichier Compose peut définir plusieurs services, et Docker Compose se charge de les démarrer et de les lier ensemble en fonction des configurations définies.

Réseaux : Docker Compose permet de définir des réseaux personnalisés pour faciliter la communication entre les conteneurs. Cela est particulièrement utile pour segmenter et gérer le trafic entre les services d'une application.

Volumes : Pour la persistance des données, Docker Compose permet de définir des volumes qui peuvent être montés dans les conteneurs, assurant que les données importantes ne sont pas perdues lors de l'arrêt ou de la destruction des conteneurs.



Utilité de Docker Compose

Simplicité et Rapidité de Déploiement : Docker Compose simplifie le déploiement d'applications composées de plusieurs services conteneurisés. Avec un seul fichier de configuration et quelques commandes, les développeurs peuvent lancer toute une application, rendant le processus rapide et reproductible.

Développement Local et Tests : Docker Compose est idéal pour le développement et les tests locaux, permettant aux développeurs de créer un environnement qui imite de près l'environnement de production, mais sur leur machine locale.

Intégration et Déploiement Continus : Les fichiers Compose peuvent être utilisés dans des pipelines CI/CD pour automatiser le déploiement d'applications. Cela facilite la mise en place de pratiques d'intégration et de déploiement continus.

Isolation des Environnements : Docker Compose permet de créer des environnements isolés pour différentes instances d'une application ou pour différents projets, évitant ainsi les conflits entre eux.

Gestion Facile des Dépendances : En définissant explicitement les services et leurs relations dans le fichier `docker-compose.yml`, Docker Compose gère automatiquement les dépendances entre les conteneurs, s'assurant que les services sont démarrés dans l'ordre correct.

Conclusion

Docker Compose est un outil essentiel pour les développeurs travaillant avec des applications Dockerisées, en particulier celles qui s'appuient sur plusieurs services interdépendants. Il offre une approche simplifiée et automatisée pour la gestion des applications conteneurisées, rendant le développement, les tests et le déploiement plus rapides et plus fiables.

Avec Docker Compose, les complexités de la configuration des services, de la gestion des réseaux et de la persistance des données sont réduites, permettant aux développeurs de se concentrer sur le développement de l'application elle-même.

Docker Swarm

Qu'est-ce que Docker Swarm ?

Docker Swarm est une fonctionnalité de clustering et d'orchestration intégrée dans Docker Engine. Elle permet de connecter plusieurs machines hôtes Docker, appelées nœuds, pour travailler ensemble comme un seul et même cluster. Grâce à Docker Swarm, vous pouvez déployer, mettre à l'échelle et gérer des conteneurs à travers plusieurs nœuds Docker de manière transparente.

Qu'est-ce qu'un cluster Docker Swarm ?

Cluster : Un groupe de machines qui exécutent Docker et sont configurées pour joindre ensemble leurs ressources. Ces machines peuvent être des serveurs physiques, des VMs, ou une combinaison des deux.

Nœuds : Chaque machine dans un Swarm est appelée un "nœud", qui peut être un "manager" ou un "worker". Les nœuds "managers" gèrent la coordination du cluster et les tâches administratives, tandis que les nœuds "workers" exécutent les conteneurs et les applications.

Services et Tâches : Dans un Swarm, vous déployez des applications sous forme de "services", qui définissent l'état désiré de l'application. Un service est essentiellement une description de la tâche à exécuter, et le Swarm s'assure que le nombre désigné de répliques de cette tâche (conteneurs) est maintenu dans le cluster.

image.png

Utilité de Docker Swarm

Haute Disponibilité : Docker Swarm assure la haute disponibilité des applications en répliquant les conteneurs sur plusieurs nœuds du cluster. Si un nœud tombe en panne, Swarm peut automatiquement redémarrer les conteneurs sur d'autres nœuds disponibles, assurant ainsi que le service reste disponible.

Équilibrage de Charge : Swarm offre des fonctionnalités d'équilibrage de charge, permettant de distribuer les requêtes entrantes entre les différentes instances (répliques) d'un service, améliorant la performance et la disponibilité de l'application.

Mise à l'échelle : Avec Docker Swarm, vous pouvez facilement mettre à l'échelle les services en augmentant ou en diminuant le nombre de répliques d'un conteneur, sans interruption de service.

Cela permet d'adapter les ressources utilisées en fonction de la demande.

Déploiement et Mise à Jour Sans Interruption : Swarm permet de déployer et de mettre à jour des applications sans temps d'arrêt grâce aux mises à jour progressives (rolling updates). Vous pouvez spécifier comment et quand les conteneurs sont mis à jour, en s'assurant que le service reste disponible pendant la mise à jour.

Découverte de Service et Réseau Overlay : Swarm fournit un réseau overlay qui connecte les conteneurs sur différents nœuds, permettant la communication inter-conteneurs comme s'ils étaient sur la même machine. La découverte de service intégrée permet aux conteneurs de se localiser et de communiquer par leurs noms de service.

Exemples

Dans ce scénario, notre client souhaite développer un site internet utilisant WordPress.

Pour stocker les données du site, nous utiliserons un conteneur MySQL. Le client aura la flexibilité de démarrer des conteneurs individuellement ou de créer une stack regroupant plusieurs conteneurs interconnectés, selon ses besoins spécifiques.

Sans-titre-2024-03-11-1845.png

Dans ce deuxième scénario, pour réduire les risques de panne, le client met en place un cluster Docker. Il déploie ensuite un "service", un mécanisme qui facilite la réplication de conteneurs en appliquant diverses contraintes adaptées aux exigences du projet.

L'objectif est de minimiser les risques de défaillance ou d'interruption du service de l'application. Cette configuration offre aussi l'avantage de simplifier les mises à jour de l'application sans nécessiter d'arrêts de production, garantissant ainsi une continuité du service.

Sans-titre-2024-03-11-184445.png

Dans ce troisième scénario, afin d'augmenter la scalabilité et la résilience du service, les données persistantes, telles que les fichiers de configuration de WordPress et sa base de données, ne sont plus hébergées localement sur chaque nœud du cluster.

À la place, elles sont centralisées sur une autre machine virtuelle en utilisant le protocole NFS (Network File System). Cette approche réduit les contraintes associées au redéploiement d'un service et facilite grandement la gestion des sauvegardes, contribuant ainsi à une meilleure efficacité et fiabilité du système.

Sans-titre-2024-03-11-18fgyujhh45.png

Dans ce quatrième scénario, le client décide d'ajouter une deuxième machine virtuelle spécifiquement destinée au stockage des données pour renforcer la résilience et la disponibilité du système.

1. Stockage Dédié : Une deuxième machine virtuelle est introduite pour servir exclusivement de stockage des données. Cette approche sépare le traitement des données de leur stockage, ce qui facilite la gestion et la sécurisation des données.

2. Réplication des Données : Les données présentes sur le premier espace de stockage (data store 1) sont répliquées sur le second espace de stockage (data store 2) en utilisant `RSYNC`, un outil de synchronisation de fichiers. Cette réplication assure que les deux emplacements de stockage contiennent des copies identiques des données, ce qui augmente la sécurité des données en fournissant une redondance en cas de défaillance d'un des systèmes de stockage.

3. Node de Secours : Le troisième nœud du cluster est configuré comme un nœud de secours. Ce nœud peut prendre le relais en cas de défaillance des nœuds principaux, assurant ainsi la continuité du service sans interruption majeure. Ce nœud de secours peut également être intégré dans un Plan de Continuité d'Activité (PCA), garantissant que les services peuvent être rapidement restaurés ou maintenus en cas de sinistre ou de panne critique.

Cette configuration avancée permet non seulement d'assurer la disponibilité et la sécurité des données mais aussi d'améliorer la capacité du système à se rétablir rapidement après un incident.

En séparant les rôles de traitement et de stockage des données et en mettant en place une stratégie de réplication et de secours, le client crée un environnement robuste, capable de supporter des conditions d'utilisation variées et des incidents inattendus sans compromettre l'intégrité des données ou la disponibilité du service.

Sans-titre-2024-03-11-1XXX.png