# PODMAN X DOCKER

🦭 une formation présenté par Andromed.

Appuyez sur espace pour la page suivante →

LinkedIn @jimmylan

# Jimmylan Surquin

Fondateur Andromed

- Lille, France 🇫🇷
- J'écris de temps en temps pour dev.to/@jimmylansrq
- Création de contenu sur jimmylansrq
- Blog & Portfolio jimmylan.fr

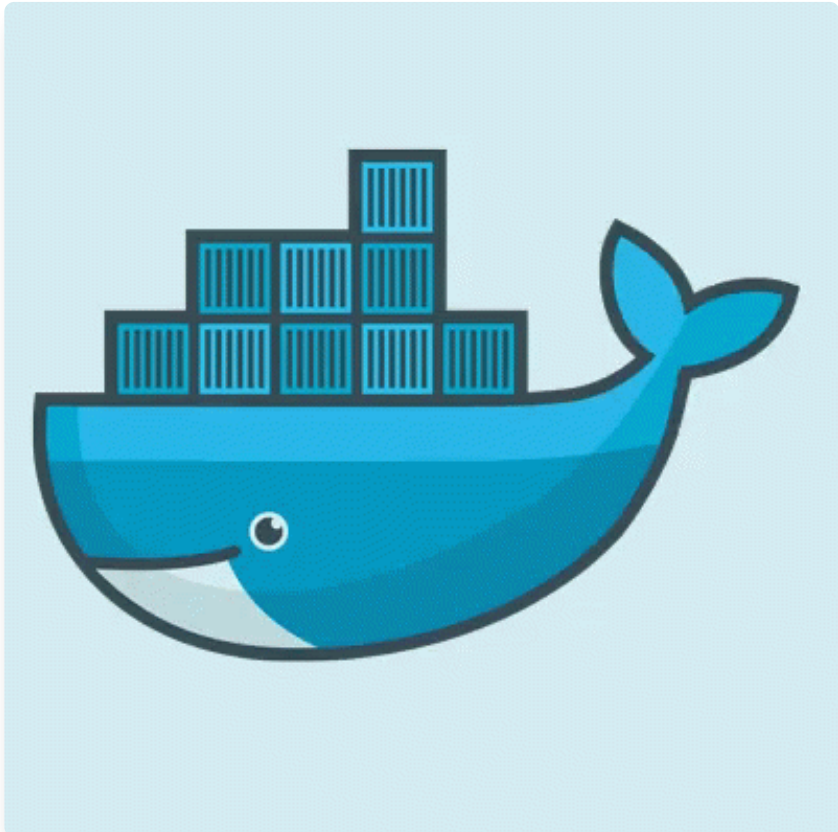
LinkedIn @jimmylan

# DISCLAIMER 🐧

## Dans cette formation nous allons voir les commandes principales de Podman.

> *Cependant les commandes sont similaires à celles de Docker.*

Je vais donc mixer les commandes de Podman et de Docker pour vous faire voir les différences mais aussi comment passer d'une commande à l'autre.



DOCKER X PODMAN

# SOMMAIRE 📜

## Voici le sommaire de cette formation sur Podman:

# Comprendre le CI/CD

Mais avant tout ! nous devons comprendre ce qu'est le CI/CD ainsi que son utilité et nous verrons en suite les microservices.

Le CI/CD est un processus qui permet de créer, tester et déployer des applications de manière automatisée.

Une métaphore pour comprendre :

- **CI** : Imaginez que vous êtes un chef cuisinier. Vous avez une recette pour faire un gâteau. Le CI (Intégration Continue) consiste à vérifier chaque ingrédient et chaque étape de la recette au fur et à mesure que vous les ajoutez, pour s'assurer que tout est correct et que le gâteau sera réussi.

- **CD** : Une fois que tous les ingrédients sont vérifiés et que la recette est prête, le CD (Déploiement Continu) consiste à mettre le gâteau au four et à le cuire automatiquement sans intervention supplémentaire, garantissant ainsi que le gâteau sera prêt à être servi dès qu'il est cuit.

# Utiliser des pipelines CI/CD

Un pipeline CI/CD est un processus qui permet de créer, tester et déployer des applications de manière automatisée.

Que veux dire pipeline ?

> Un pipeline est un processus qui permet de créer, tester et déployer des applications de manière automatisée.

> En clair : c'est tout la chaine de déploiement / le processus de déploiement.

# Pourquoi parler de CI/CD avec Docker et Podman ?

Docker et Podman sont des outils de conteneurisation.

Donc il est important de comprendre le concept de CI/CD avec ces outils.

Imaginons que nous développons une application web.

Nous voulons déployer notre application.

Nous pourrions utiliser un pipeline CI/CD pour déployer notre application.

À chaque push sur le dépôt git, nous voulons déployer notre application.

Nous pourrions utiliser un pipeline CI/CD pour déployer notre application.

Donc lancer le docker-compose ou le dockerfile au besoin. (mais nous y reviendrons)

# Comprendre les micro-services

Un micro-service est une application indépendante qui peut être créée à partir d'un système d'exploitation ou d'un environnement logiciel spécifique.

Une métaphore pour comprendre :

- **Micro-service** : Imaginez que vous êtes dans un supermarché. Chaque rayon est un micro-service qui gère un type de produit spécifique. Par exemple, le rayon des fruits s'occupe uniquement des fruits, tandis que le rayon des produits laitiers s'occupe uniquement des produits laitiers. Chaque rayon fonctionne de manière indépendante mais contribue à l'ensemble du supermarché.

# Pourquoi utiliser les micro-services ?

Les micro-services permettent de découper une application en plusieurs services indépendants qui peuvent être développés, déployés et gérés de manière indépendante. Cela permet de rendre l'application plus modulaire, plus facile à maintenir et plus scalable.

# Exemple concret et … à quoi ça sert ?

Imaginons que nous développons une application de e-commerce.

Nous pourrions avoir les micro-services suivants :

- **Microservice de gestion de produits** : Gère la gestion des produits, les stocks, les prix, etc.
- **Microservice de gestion de commandes** : Gère la gestion des commandes, la facturation, la livraison, etc.
- **Microservice de gestion de paiement** : Gère la gestion des paiements, les transactions, etc.
- **Microservice de gestion des utilisateurs** : Gère la gestion des utilisateurs, les comptes, les permissions, etc.

J'espère que vous avez compris le concept de micro-service.

Car à vrai dire quand vous utilisez podman/docker vous n'allez pas créer des micro-services.

Mais vous allez utiliser des containers qui eux même peuvent être des micro-services.

Et de toute façon cette architecture est utilisé dans la vie de tous les jours.

# QCM sur les micro-services et le CI/CD

## 1. Quel est l'avantage principal des micro-services ?

- ☐ Ils permettent de créer des applications monolithiques.
- ☐ Ils permettent de découper une application en plusieurs services indépendants.
- ☐ Ils nécessitent moins de ressources que les applications traditionnelles.
- ☐ Ils sont plus difficiles à maintenir.

## 2. Dans l'exemple d'une application de e-commerce, quel micro-service gère les transactions de paiement ?

- ☐ Microservice de gestion de produits
- ☐ Microservice de gestion de commandes
- ☐ Microservice de gestion de paiement
- ☐ Microservice de gestion des utilisateurs

# 3. Pourquoi utiliser les micro-services ?

- ☐ Pour rendre l'application plus modulaire, plus facile à maintenir et plus scalable.
- ☐ Pour augmenter la complexité de l'application.
- ☐ Pour réduire le nombre de développeurs nécessaires.
- ☐ Pour éviter l'utilisation de conteneurs.

## 4. Quel est l'objectif principal du CI/CD ?

- ☐ Augmenter la complexité du développement logiciel.
- ☐ Automatiser le processus de développement, de test et de déploiement.
- ☐ Réduire la qualité du code.
- ☐ Remplacer les développeurs par des machines.

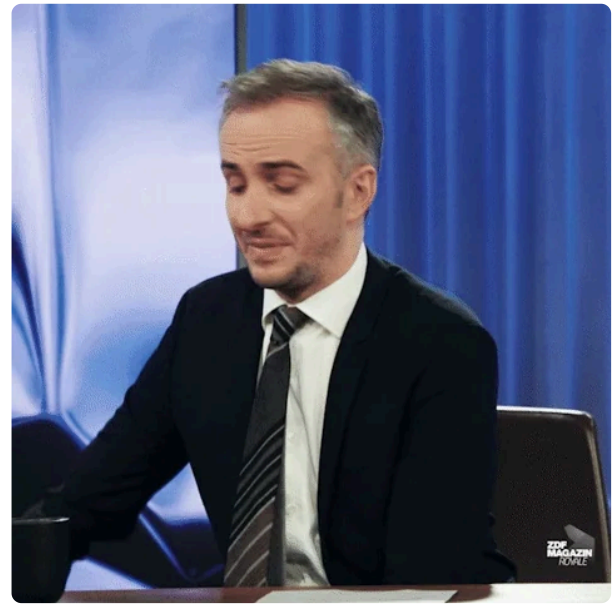
## 5. Quel outil est couramment utilisé pour le CI/CD ?

- ☐ Docker Hub
- ☐ Jenkins
- ☐ GitHub Packages
- ☐ Quay.io

# Réponse(s)

1. Ils permettent de créer des applications modulaires et indépendantes.
2. Microservice de gestion de paiement
3. Pour rendre l'application plus modulaire, plus facile à maintenir et plus scalable.
4. Automatiser le processus de développement, de test et de déploiement.
5. GitHub Packages / Jenkins

# Des définitions avant tout



# Définition de virtualisation

La virtualisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans ce qu'on appelle une machine virtuelle.

# Définition de conteneur

Un conteneur est un environnement isolé qui permet de déployer des applications à partir d'un système d'exploitation ou d'un environnement logiciel.

# Définition de conteneurisation

La conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.

# Définition de machine virtuelle

Une machine virtuelle est un environnement logiciel qui permet d'exécuter des systèmes d'exploitation ou des applications de manière isolée, en simulant un matériel informatique.

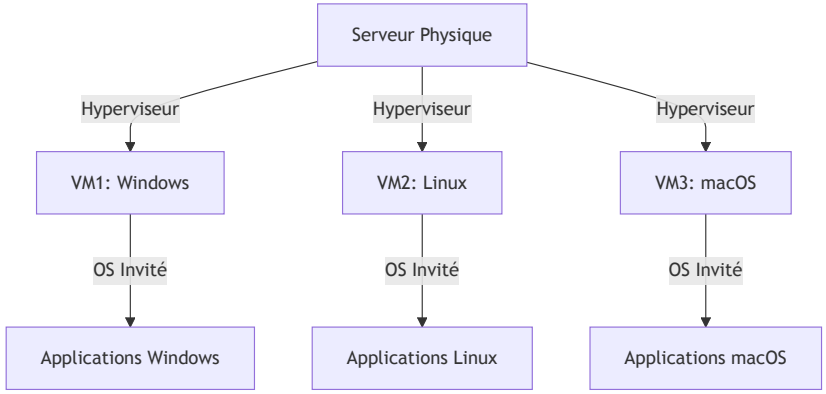
# Virtualisation vs conteneurisation

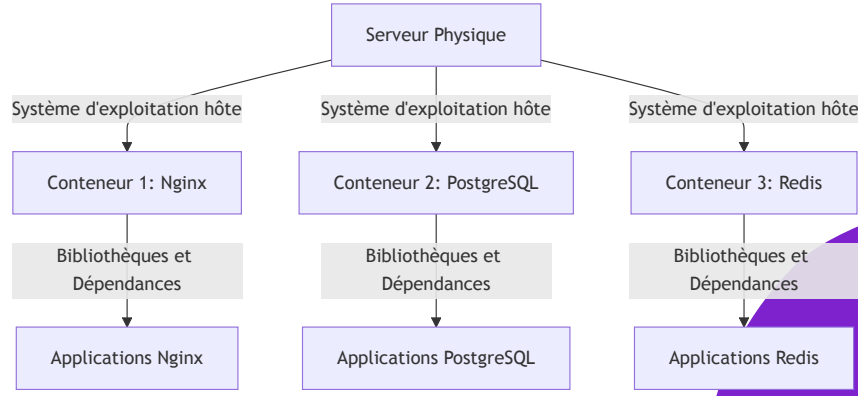
La virtualisation et la conteneurisation sont deux concepts liés à la gestion des ressources informatiques.

- **Virtualisation** : La virtualisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans un conteneur.
- **Conteneurisation** : La conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.

# Schéma de la Virtualisation et de la Conteneurisation

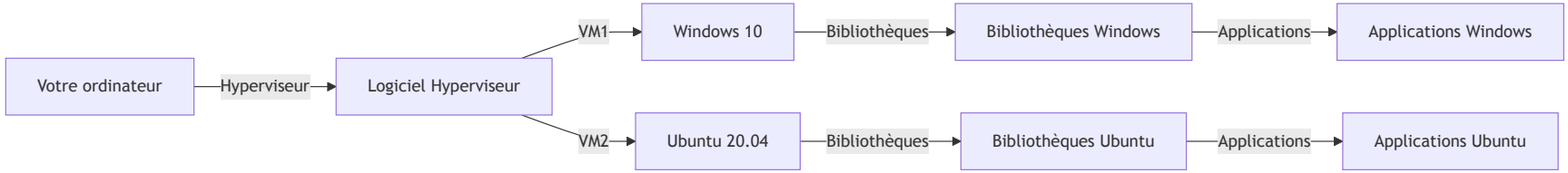
Voici un schéma qui illustre les différences entre la virtualisation et la conteneurisation.



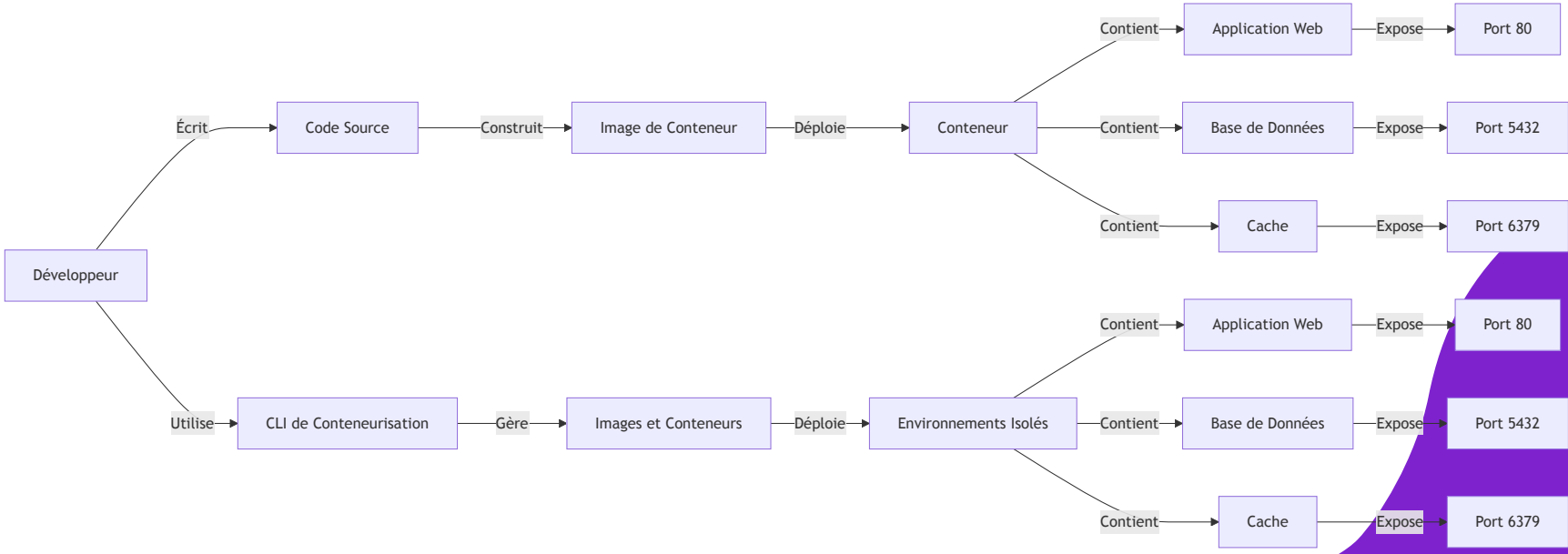
# Comment fonctionne la virtualisation ?

La virtualisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans un conteneur.



# Comment fonctionne la conteneurisation ?

La conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.



# Définition de Kernel

Le kernel est le cœur du système d'exploitation qui gère les ressources matérielles et les interactions entre le matériel et les logiciels. Les conteneurs sont des environnements isolés qui partagent le même kernel mais fonctionnent indépendamment les uns des autres.

## Une petite façon simple de comprendre le kernel :

Le kernel est le cerveau du système d'exploitation.

C'est lui qui gère les ressources matérielles et les interactions entre le matériel et les logiciels.

# QCM sur les définitions

## Qu'est ce que la virtualisation ?

- ☐ La virtualisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans un conteneur.
- ☐ La virtualisation est un processus qui permet de créer une machine virtuelle à partir d'un système d'exploitation ou d'un environnement logiciel.
- ☐ La virtualisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.
- ☐ La virtualisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.

# Qu'est ce que la conteneurisation ?

- ☐ La conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.
- ☐ La conteneurisation est un processus qui permet de créer une machine virtuelle à partir d'un système d'exploitation ou d'un environnement logiciel.
- ☐ La conteneurisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans un conteneur.
- ☐ La conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.

# Quel est la différence entre la virtualisation et la conteneurisation ?

- ☐ La virtualisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans un conteneur alors que la conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.
- ☐ La virtualisation est un processus qui permet de créer une machine virtuelle à partir d'un système d'exploitation ou d'un environnement logiciel alors que la conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel.
- ☐ La conteneurisation est un processus qui permet de créer un conteneur à partir d'un système d'exploitation ou d'un environnement logiciel alors que la virtualisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans un conteneur.
- ☐ La conteneurisation est un processus qui permet de créer une image d'un système d'exploitation ou d'un environnement logiciel dans un conteneur.


# Introduction à Podman

Podman est un outil de gestion de conteneurs qui permet de créer, gérer et exécuter des conteneurs sans nécessiter de démon.

Il est compatible avec les commandes Docker, ce qui facilite la transition pour les utilisateurs de Docker.

Podman offre également des fonctionnalités supplémentaires telles que la gestion des pods et une meilleure sécurité grâce à son architecture sans démon (mais nous en reparlerons plus tard dans cette formation).

# Différences entre Docker et Podman

| Fonctionnalité | Docker | Podman |
|---|---|---|
| **Démon** | Nécessite un démon pour fonctionner | N'a pas besoin de démon pour fonctionner |
| **Pods** | Ne gère pas les pods | Offre une gestion des pods, permettant de regrouper plusieurs conteneurs |
| **Sécurité** | Fonctionne avec un démon, ce qui peut poser des problèmes de sécurité | Conçu pour une meilleure sécurité grâce à son architecture sans démon |
| **Compatibilité** | - | Commandes compatibles avec celles de Docker, facilitant la transition |
| **Rootless** | Nécessite des privilèges root pour certaines opérations | Permet l'exécution de conteneurs en tant qu'utilisateur non root par défaut |
| **Outils standards** | Utilise des outils spécifiques à Docker | Utilise des outils standards de Linux pour la gestion des conteneurs, comme systemd |
| **Images** | Nécessite un démon d'arrière-plan pour créer des images | Permet de créer des images de conteneurs sans nécessiter de démon d'arrière-plan |

# Explication et définition d'un démon

Un démon est un programme qui s'exécute en arrière-plan et qui gère les ressources du système.

# Explication d'un pod

Un pod est un groupe de conteneurs qui partagent le même espace de réseau et qui sont déployés ensemble.

# Qu'est ce que root / rootless ?

- **Root** : Un utilisateur avec des privilèges root peut faire tout ce que bon lui semble dans le système.
- **Rootless** : Un utilisateur non root ne peut pas faire des choses comme installer des paquets, modifier des fichiers systèmes, etc.

# Le CLI Docker/Podman

Nous allons voir les commandes principales de Docker.

## Commandes principales Docker

| Commande | Description |
| --- | --- |
| `docker/podman run` | Exécute une commande dans un nouveau conteneur |
| `docker/podman ps` | Liste les conteneurs en cours d'exécution |
| `docker/podman stop` | Arrête un conteneur en cours d'exécution |
| `docker/podman rm` | Supprime un conteneur arrêté |

| Commande | Description |
| --- | --- |
| `docker/podman pull` | Télécharge une image depuis un registre |
| `docker/podman images` | Liste les images disponibles localement |
| `docker/podman rmi` | Supprime une ou plusieurs images |
| `docker/podman exec` | Exécute une commande dans un conteneur en cours d'exécution |
| `docker/podman build` | Construit une image à partir d'un Dockerfile |
| `docker/podman push` | Envoie une image à un registre |
| `docker/podman tag` | Ajoute un tag à une image |
| `docker/podman login` | Connecte à un registre |
| `docker/podman logout` | Déconnecte d'un registre |

# Commandes avancées

Voici quelques commandes avancées.

## Commandes avancées

| Commande | Description |
| --- | --- |
| `docker/podman network create` | Crée un nouveau réseau Docker |
| `docker/podman volume create` | Crée un nouveau volume Docker |
| `docker/podman inspect` | Affiche les détails d'un conteneur ou d'une image |
| `docker/podman logs` | Affiche les logs d'un conteneur |

# Encore des Commandes avancées

Voici quelques autres commandes Docker avancées.

## Commandes avancées supplémentaires

| Commande | Description |
| --- | --- |
| `docker/podman-compose up` | Démarre et attache des conteneurs définis dans un fichier docker-compose |
| `docker/podman-compose down` | Arrête et supprime les conteneurs, réseaux, volumes définis dans un fichier docker-compose |
| `docker/podman-compose logs` | Affiche les logs des services définis dans un fichier docker-compose |
| `docker/podman-compose exec` | Exécute une commande dans un conteneur en cours d'exécution défini dans un fichier docker-compose |

Petite astuce :

Depuis la version 2.0 de docker , vous n'êtes plus obligé d'écrire docker-compose avec le tiret du milieu mais vous pouvez faire :

```
docker compose up
```

## Directement !

# Images Podman

Déjà reprenons ce qu'est une image.

Une image est un fichier qui contient un système d'exploitation ou un environnement logiciel.

**Exemple :**

```
docker pull ubuntu:latest
```

Cela va nous donner une image de la distribution linux Ubuntu.

Je vais donc **À PARTIR DE CETTE IMAGE** créer un **CONTENEUR**.

Je peux récuperer des images sur des registres comme :

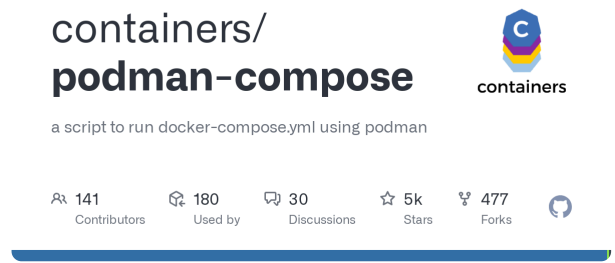
- Docker Hub
- Quay.io
- GitHub Packages
- etc

Mais bien sur je peux aussi créer mes images.
(sois à partir de rien, sois à partir d'une autre image dont je vais créer des surcouches)

# Astuce en plus, on peut directement chercher des images avec la commande :

```
docker search <image>
```

# Créer son premier conteneur



# Créer son premier conteneur

```
podman run -d --name my-container -p 8080:80 nginx
```

## Explications

- `podman run` : Commande pour créer et exécuter un conteneur.
- `-d` : Exécute le conteneur en arrière-plan.
- `--name my-container` : Nom du conteneur.
- `-p 8080:80` : Port du conteneur. (8080 sur l'host, 80 dans le conteneur)
- `nginx` : Image à utiliser.

# Petit exercice :

Créer un conteneur qui tourne une image nginx ou de votre choix et qui est accessible sur votre host.

Vous avez déjà tout ce qu'il faut dans la slide précédente.

# PodmanFile

podman

# En premier, définition d'un Dockerfile/PodmanFile.

Un Dockerfile est un fichier qui contient les instructions pour créer une image de conteneur.

Un podmanFile pareil, mais pour podman.

**L'idée, est de pouvoir créer des images de conteneurs de manière custom.** (car je le rappel, on peut aussi utiliser des images officielle comme ubuntu, debian, etc..., mais forcement, elle ne seront pas personnalisée à mon application mais juste une image de base)

Par exemple, si j'ai besoin d'une image avec une version de node spécifique, je peux créer une image avec la version de node et toutes les dépendances dont j'ai besoin.

# Pour l'utiliser, il faut faire :

```
podman build -t my-image .
```

Explications :

- `podman build` : Commande pour créer une image à partir d'un Dockerfile.
- `-t my-image` : Nom de l'image.
- `.` : Répertoire où se trouve le Dockerfile, ici à la racine du projet.

# Petit exercice :

Créer un Dockerfile/PodmanFile qui permet de créer une image avec une version de node spécifique et qui est accessible sur votre host.

# Un mauvais dockerfile

```
 # Utilisation d'une image de base lourde et non nécessaire pour l'application
FROM ubuntu:latest

# Ne pas spécifier de mainteneur - manque de clarté sur qui a créé cette image
MAINTAINER "someone@example.com"

# Exécution d'une seule commande apt-get sans update, peut conduire à des paquets obsolètes ou vulnéral
RUN apt-get install -y curl
```

ps : suite sur la deuxieme slide

```
 # Le code de l'application est copié avant d'installer les dépendances, ce qui casse la mise en cache
COPY . /app

# Exécution de plusieurs commandes RUN dans une seule instruction, rendant difficile le débogage et la
RUN cd /app && \
    mkdir temp && \
    touch temp/file.txt && \
    echo "Creating a temporary file"

# Mauvais usage de l'utilisateur root, les applications ne devraient pas tourner avec ces privilèges pa
USER root

# Utilisation d'un port non nécessaire pour l'application
EXPOSE 1234
```

# Explication des erreurs :

1. **FROM ubuntu:latest** : L'image Ubuntu est lourde pour la plupart des applications, préférer une image plus légère comme Alpine ou une image spécifique à l'environnement d'exécution (par exemple, `node:alpine` , `python:slim` ). De plus, utiliser `:latest` peut introduire des problèmes de version instable, mieux vaut utiliser une version spécifique.

2. **MAINTAINER** : Cette instruction est obsolète dans les versions récentes de Docker. Utilisez `LABEL maintainer="someone@example.com"` à la place.

3. **RUN apt-get install -y curl** : Il manque une commande `apt-get update` avant l'installation des paquets, ce qui peut entraîner des paquets obsolètes. De plus, l'installation de `curl` pourrait ne pas être nécessaire, cela ajoute du poids à l'image inutilement.

4. **COPY . /app** : Le code est copié avant d'installer les dépendances, ce qui casse la mise en cache de Docker. Pour une meilleure optimisation, les dépendances doivent être installées avant de copier l'ensemble du code source, surtout si elles sont rarement modifiées.

5. **RUN cd /app && \ mkdir temp && \ touch temp/file.txt** : Il y a plusieurs commandes dans une seule instruction `RUN` , ce qui rend le débogage difficile. Si une seule partie échoue, il sera compliqué d'identifier laquelle. En plus, la création d'un fichier temporaire dans une étape de build n'a aucun sens si l'application ne l'utilise pas directement.

6. **USER root** : Utiliser l'utilisateur root pour exécuter des applications n'est pas recommandé pour des raisons de sécurité. Il vaut mieux créer un utilisateur non privilégié et l'utiliser pour exécuter l'application.

7. **EXPOSE 1234** : Exposer un port qui n'est pas utilisé par l'application est inutile et peut prêter à confusion.

8. **CMD ["echo", "Hello World"]** : Cette commande ne démarre pas réellement une application. Elle ne fait qu'afficher un message, ce qui ne reflète pas le comportement attendu pour une application Docker.

# Un bon dockerfile

Voyons ici un bon Dockerfile.

```
 # Utilisation d'une image de base légère et adaptée à l'application
FROM alpine:3.16

# Déclaration du mainteneur via l'instruction LABEL (plus moderne que MAINTAINER)
LABEL maintainer="someone@example.com"

# Mise à jour des paquets et installation de curl proprement
# Combine apt-get update et install pour réduire les couches et garder l'image à jour
RUN apk update && apk add --no-cache curl

# Installation des dépendances avant de copier le code source pour optimiser le cache Docker
# Cela garantit que les dépendances sont réutilisées si le code source change
WORKDIR /app
```

ps : suite sur la deuxieme slide

```
 # Copie du fichier de dépendances uniquement (si applicable, par ex: package.json pour Node.js, requir
# COPY package.json /app  <-- Exemple de bonne pratique pour Node.js ou Python

# Installation des dépendances (si applicable)
# RUN npm install ou pip install -r requirements.txt

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

# Création d'un utilisateur non root pour éviter les risques de sécurité liés à l'exécution en tant que
RUN adduser -D -g '' appuser
USER appuser

# Exposer uniquement le port nécessaire par l'application
EXPOSE 8080
```

# Pouquoi est-ce un bon Dockerfile ?

1. **FROM alpine:3.16** : Alpine est une image de base très légère (seulement quelques Mo) par rapport à Ubuntu, ce qui réduit la taille globale de l'image Docker. En spécifiant une version précise ( `3.16` ), on garantit la stabilité.

2. **LABEL maintainer="someone@example.com"** : La commande `LABEL` est la méthode recommandée pour spécifier le mainteneur de l'image, car elle est plus moderne et flexible que l'ancienne instruction `MAINTAINER` .

3. **RUN apk update && apk add --no-cache curl** : L'utilisation de `apk update` permet de s'assurer que les paquets sont à jour avant l'installation. L'option `--no-cache` évite de stocker des fichiers temporaires inutiles, ce qui optimise l'image en la rendant plus petite.

4. **WORKDIR /app** : `WORKDIR` définit le répertoire de travail où toutes les actions suivantes auront lieu, au lieu d'utiliser des commandes `cd` . C'est plus propre et plus lisible.