

# Docker

# Sommaire

- Introduction rappel sur les conteneurs
  - Fondamentaux des conteneurs : SaaS, PaaS, IaaS, modèles de déploiement, et leur différenciation avec la virtualisation.
  - Historique et acteurs du marché : De LXC à Docker, introduction à l'orchestration et aux acteurs principaux comme Red Hat, Azure.
- Déploiement et Gestion des Conteneurs avec Docker
  - Préparation, configuration, et gestion de conteneurs : bonnes pratiques et cycle de vie.

# Introduction rappel sur les conteneurs

## 1. Fondamentaux des conteneurs

- **Conteneurs:** Les conteneurs sont des unités standardisées de logiciels qui regroupent le code et toutes ses dépendances afin que l'application puisse s'exécuter rapidement et de manière fiable d'un environnement informatique à un autre.
  - **Isolation :** Les conteneurs partagent le même système d'exploitation hôte mais s'exécutent dans des espaces utilisateurs isolés, tandis que la virtualisation utilise des hyperviseurs pour exécuter plusieurs systèmes d'exploitation séparés.
  - **Performance :** Les conteneurs sont plus légers et ont pratiquement pas de surcharge liée à l'hyperviseur, ce qui permet un démarrage quasi instantané et une meilleure utilisation des ressources.
- **SaaS, PaaS, IaaS:** Ces trois modèles de services cloud représentent différentes couches d'abstraction :
  - **Infrastructure as a Service (IaaS) :** Fournit des ressources informatiques virtualisées sur Internet (ex. AWS EC2).
  - **Platform as a Service (PaaS) :** Offre un environnement de développement et de déploiement d'applications sans la complexité de la gestion des infrastructures sous-jacentes (ex. Heroku).
  - **Software as a Service (SaaS) :** Distribue des logiciels accessibles via Internet, gérés entièrement par des tiers (ex. Google Workspace).

# Introduction rappel sur les conteneurs

## 2. Historique et acteurs du marché: De LXC à Docker

- **Linux Containers (LXC)** : Précurseur dans la technologie des conteneurs, LXC était basé sur la virtualisation au niveau du système d'exploitation, utilisant des cgroups et des espaces de noms du noyau Linux pour isoler les processus.
- **Docker** : Lancé en 2013, Docker a révolutionné le domaine en simplifiant la création et la gestion des conteneurs grâce à des commandes simples et un format de conteneur standard.
- **Orchestration**: Avec l'augmentation de l'utilisation des conteneurs, la nécessité de gérer de grandes flottes de ceux-ci a conduit au développement de systèmes d'orchestration tels que Kubernetes, qui automatise le déploiement, la mise à l'échelle, et la gestion des applications conteneurisées.

# Déploiement et gestion des conteneurs avec docker

## 1. Préparation, configuration, et gestion de conteneurs

- **Préparation et configuration**

- **Installation de docker:** Pour préparer votre environnement à l'utilisation de Docker, vous devez d'abord installer Docker Desktop ou Docker Engine, selon que vous êtes sur un système d'exploitation Windows, Mac ou Linux. L'installation implique généralement le téléchargement du package approprié et son installation en suivant les instructions spécifiques à la plateforme.
- **Configuration de docker:** Une fois Docker installé, vous pouvez ajuster les paramètres via le Docker Desktop ou directement dans les fichiers de configuration de Docker. Ces paramètres peuvent inclure des allocations de ressources (comme la mémoire et le CPU) pour optimiser les performances des conteneurs.

## 2. Cycle de vie des conteneurs

- **Développement:** L'environnement de développement, où le code est écrit et testé dans des conteneurs locaux pour assurer la cohérence entre les environnements de développement, de test et de production.
- **Intégration et déploiement:** Les conteneurs sont intégrés dans des pipelines CI/CD, où des tests automatisés sont exécutés et les images de conteneurs sont construites et déployées.
- **Exécution et surveillance:** En production, les conteneurs sont surveillés pour leur performance et leur santé, utilisant des outils comme Docker Swarm ou Kubernetes pour gérer le déploiement à grande échelle.

# Déploiement et gestion des conteneurs avec docker

## 3. Création et gestion d'images de conteneurs, réseau Docker

- **Création d'Images**

- **Dockerfile:** Au cœur de la création d'image se trouve le Dockerfile. Il commence généralement avec une instruction `FROM` pour définir l'image de base. Ensuite, des instructions comme `RUN`, `COPY`, et `ADD` permettent d'ajouter des fichiers, exécuter des commandes et installer des logiciels.
- **Construire l'image:** Utilisez `docker build -t my-app:version .` pour construire l'image à partir du Dockerfile. Ce processus convertit le Dockerfile en une série de couches d'images, où chaque instruction crée une nouvelle couche.

```
# Définir l'image de base
FROM ubuntu:20.04

# Installer nginx
RUN apt-get update && apt-get install -y nginx

# Ajouter les fichiers de l'application
COPY ./html /var/www/html

# Exposer le port 80
EXPOSE 80

# Commande pour démarrer le serveur web
CMD ["nginx", "-g", "daemon off;"]
```

# Déploiement et gestion des conteneurs avec docker

- **Gestion d'images**

- **Gestion des images locales:** La commande `docker images` liste toutes les images locales. Les images peuvent être supprimées avec `docker rmi` ou tagguées différemment pour aider à leur organisation.
- **Registres d'images:** Pour partager et stocker des images, Docker Hub ou d'autres registres comme AWS ECR peuvent être utilisés. Utilisez `docker push` pour téléverser une image vers un registre et `docker pull` pour la télécharger.

- **Création d'images : Multi-Stage Builds**

Les builds multi-stages sont une méthode puissante pour créer des images Docker propres, optimisées, et minimales sans nécessiter de scripts ou de procédures supplémentaires.

1. **Réduction de la taille de l'image :** Les builds multi-stages permettent de séparer les étapes de construction et de production dans des images distinctes. Cela signifie que vous pouvez utiliser une image avec tous les outils nécessaires pour construire votre application et ensuite copier seulement les artefacts finaux (par exemple, les fichiers binaires ou le code compilé) dans une image de production plus légère qui ne contient pas les outils de build inutiles.
2. **Sécurité améliorée :** En minimisant la surface d'attaque de vos images de conteneur (en ne comprenant que les fichiers nécessaires à l'exécution de l'application), vous réduisez le risque associé à des dépendances et outils inutiles en production.

# Déploiement et gestion des conteneurs avec docker

## Dockerfile Multi-Stage pour une application PHP:

```
FROM php:7.4-cli as builder
WORKDIR /app
COPY . /app
RUN curl -sS https://getcomposer.org/installer | php
RUN php composer.phar install --no-dev

FROM php:7.4-cli
WORKDIR /app
COPY --from=builder /app /app
EXPOSE 80
CMD ["php", "-S", "0.0.0.0:80"]
```



# Déploiement et gestion des conteneurs avec docker

- **Les couches (Layers) dans docker images:** Les images Docker sont construites en utilisant un système de couches superposées (layers). Chaque couche représente une modification de l'image, telle qu'un fichier ajouté, modifié ou supprimé par rapport à la couche précédente. Voici quelques points clés sur ce fonctionnement :
1. **Immutabilité** : Chaque couche est immuable, c'est-à-dire qu'une fois créée, elle ne peut plus être modifiée, seulement remplacée ou supprimée. Cette immutabilité assure que l'image est constante et prévisible à chaque déploiement.
  2. **Réutilisation** : Lors de la construction de nouvelles images, Docker cherche à réutiliser les couches existantes. Si plusieurs images sont construites à partir de la même base ou partagent des instructions similaires dans leurs Dockerfiles (par exemple, le même système d'exploitation ou les mêmes packages installés), elles partageront les mêmes couches, réduisant ainsi l'espace de stockage nécessaire et accélérant les processus de build et de déploiement.
  3. **Effet sur la taille de l'image** : Sans optimisation, chaque nouvelle instruction dans un Dockerfile peut ajouter une nouvelle couche, augmentant potentiellement la taille de l'image. Cela peut inclure des données inutiles qui ne sont pas nécessaires pour exécuter l'application (comme les outils de build ou les fichiers temporaires).

# Déploiement et gestion des conteneurs avec docker

- **Multi-Stage builds et l'importance des layers** : Les builds multi-stages sont particulièrement utiles pour exploiter efficacement le système de couches de Docker
1. **Séparation des environnements de build et de runtime** : Vous pouvez utiliser un premier stage avec une image de base contenant tous les outils nécessaires à la construction de l'application (comme les compilateurs, les dépendances de build, etc.). Après que l'application est construite, vous n'avez besoin que des artefacts de build (exécutables, fichiers compilés) pour l'exécution.
  2. **Réduction de la taille finale de l'image** : En passant au deuxième stage pour l'image de production, vous commencez avec une nouvelle image de base, souvent beaucoup plus petite, et n'y copiez que les artefacts nécessaires du premier stage. Cela signifie que toutes les couches contenant les outils et les fichiers temporaires ne sont pas incluses dans l'image finale.

## Exercice

- Créez un dockerfile pour déployer l'application située dans le dépôt git "<https://github.com/NouvelleTechno/e-commerce-Symfony-6?tab=readme-ov-file>"

# Réseau Docker

## 1. Bridge (Pont)

### Fonctionnement :

- Chaque réseau bridge crée une interface réseau virtuelle sur l'hôte, qui fait office de passerelle pour les conteneurs connectés à ce réseau.
- Les conteneurs sur un même réseau bridge peuvent communiquer entre eux, et le NAT (Network Address Translation) est utilisé pour connecter ces conteneurs à l'extérieur.

```
# Création d'un réseau bridge personnalisé
docker network create --driver bridge mon_bridge

# Exécution d'un conteneur sur ce réseau
docker run --network mon_bridge --name mon_conteneur_1 nginx

# Exécution d'un deuxième conteneur sur le même réseau
docker run --network mon_bridge --name mon_conteneur_2 nginx
```

# Réseau Docker

## 2. Host (Hôte)

### Fonctionnement :

- En utilisant le pilote de réseau host, les conteneurs utilisent directement l'interface réseau de l'hôte, ce qui élimine le surcoût de la virtualisation réseau et améliore les performances.

```
# Exécution d'un conteneur utilisant le réseau de l'hôte
docker run --network host --name mon_conteneur_host nginx
```

## 3. Overlay (Superposition)

### Fonctionnement :

- Les réseaux overlay utilisent des technologies de réseau virtuel pour permettre la communication entre des conteneurs s'exécutant sur différents hôtes Docker.
- Ce type de réseau est souvent utilisé avec Docker Swarm pour gérer un cluster de conteneurs.

```
# Création d'un réseau overlay (nécessite Docker Swarm)
docker network create --driver overlay mon_overlay

# Exécution d'un service sur ce réseau
docker service create --name mon_service --network mon_overlay nginx
```

# Réseau Docker

## 3. Macvlan

### Fonctionnement :

- Le pilote macvlan crée une interface MAC virtuelle pour chaque conteneur, leur permettant de paraître comme des dispositifs physiques distincts sur le réseau, ce qui est utile pour des applications nécessitant une interaction directe avec le réseau physique sans passer par un NAT.

```
# Création d'un réseau macvlan
docker network create -d macvlan --subnet=192.168.1.0/24 --gateway=192.168.1.1 -o parent=eth0 mon_macvlan

# Exécution d'un conteneur sur ce réseau
docker run --network mon_macvlan --name mon_conteneur_macvlan nginx
```

# Réseau Docker DNS et Découverte de Service

## Fonctionnement :

- Docker maintient un service DNS interne pour les réseaux personnalisés, ce qui permet aux conteneurs de se référer les uns aux autres par leurs noms.

## Exemple de découverte de service :

```
# Création d'un réseau personnalisé
docker network create --driver bridge mon_reseau_dns

# Exécution de deux conteneurs sur ce réseau
docker run --network mon_reseau_dns --name serveur nginx
docker run --network mon_reseau_dns --name client busybox ping serveur
```

## TP

**Sujet du TP:** Déploiement d'une application web PHP existant avec MySQL et Nginx en utilisant Docker

**Objectif:** Déployer une application PHP existante en utilisant Docker, en intégrant une base de données MySQL et en configurant Nginx comme reverse proxy pour gérer les requêtes. Les stagiaires pratiqueront la création de réseaux Docker, le déploiement de conteneurs et la configuration de la communication inter-conteneurs.



**Merci pour votre attention**

**Des questions ?**