UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Docker

Utopios® Tous droits réservés

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.

Utopios® Tous droits réservés

Sommaire

- Stockage et Réseau dans Docker
  - Gestion avancée du stockage : volumes Docker et persistance des données.
  - Atelier pratique : Mise en œuvre des volumes
- Monitoring, Maintenance et Orchestration
- Mise en Œuvre de Docker Compose
  - Notion de microservices, utilisation de Docker Compose, introduction aux architectures N-tiers

Utopios® Tous droits réservés

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).

Utopios® Tous droits réservés

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.

Utopios® Tous droits réservés

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 cie 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.

Utopios® Tous droits réservés

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;"]
```

Utopios® Tous droits réservés

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.

Utopios® Tous droits réservés

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"]
```

Utopios® Tous droits réservés

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).

Utopios® Tous droits réservés

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.

Utopios® Tous droits réservés

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"

Utopios® Tous droits réservés

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
```

Utopios® Tous droits réservés

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
```

Utopios® Tous droits réservés

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
```

Utopios® Tous droits réservés

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
```

Utopios® Tous droits réservés

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.

Utopios® Tous droits réservés

Volumes Docker

Le stockage dans Docker est un aspect crucial pour la gestion des données de manière efficace et sécurisée, surtout quand on veut assurer la persistance des données au-delà du cycle de vie des conteneurs.

1. Volumes Docker

Les volumes sont la méthode recommandée par Docker pour persister des données générées et utilisées par les conteneurs Docker. Ils sont complètement gérés par Docker.

Caractéristiques :

- **Indépendants des conteneurs :** Les volumes existent indépendamment des conteneurs, ce qui signifie que Docker ne supprime pas automatiquement les volumes lorsque vous supprimez un conteneur, sauf si vous le spécifiez explicitement.
- **Support multi-hôte :** Avec des plugins de volumes comme REX-Ray ou Flocker, vous pouvez monter des volumes dans des conteneurs qui s'exécutent sur différents hôtes dans un cluster Docker.
- **Performances :** Les volumes sont généralement plus performants que les alternatives car ils permettent une gestion directe par Docker.

```
# Création d'un volume
docker volume create data-volume

# Démarrage d'un conteneur avec ce volume monté
docker run -d --name my-container -v data-volume:/usr/share/data ubuntu
```

Utopios® Tous droits réservés

Volumes Docker

2. Bind Mounts

Les bind mounts peuvent être utilisés pour stocker des données sur le système hôte, mais ils sont moins isolés que les volumes. Ils permettent aux données d'être stockées directement sur le système de fichiers de l'hôte et peuvent être utilisés pour des cas spécifiques où un accès direct aux fichiers de l'hôte est nécessaire.

Caractéristiques :

- **Contrôle précis :** Vous contrôlez l'emplacement exact sur l'hôte où les données sont stockées.
- **Accessibilité :** Les données peuvent être manipulées directement sur l'hôte.
- **Dépendant du système hôte :** Le chemin spécifié doit exister sur l'hôte.

```
# Démarrage d'un conteneur avec un bind mount
docker run -d --name my-dev-container -v /path/on/host:/path/in/container ubuntu
```

Utopios® Tous droits réservés

Volumes Docker

3. tmpfs Mounts

Pour les données qui ne doivent pas être persistantes ou sécurisées, Docker permet l'utilisation de `tmpfs` mounts. Ces derniers stockent les données en mémoire seulement, et elles sont effacées lorsque le conteneur est arrêté.

Caractéristiques :

- **Rapide :** Stockage et accès aux données très rapide car elles résident en mémoire.
- **Sécurité :** Les données ne sont pas écrites sur le disque, ce qui est utile pour traiter des informations sensibles.

```
# Démarrage d'un conteneur avec tmpfs
docker run -d --name my-temp-container --tmpfs /tmp ubuntu
```

Utopios® Tous droits réservés

TP Volumes Docker

Contexte : Lors du développement d'applications web, il est essentiel de maintenir la persistance des données à travers les cycles de développement et d'éviter la perte de données lors des redémarrages des conteneurs Docker. L'utilisation de volumes Docker offre une solution pour gérer la persistance des données efficacement.

Objectif : Configurer un environnement de développement pour une application Symfony qui utilise Docker. L'accent sera mis sur la création et la gestion de volumes Docker pour assurer la persistance des données de l'application et de la base de données.

Utopios® Tous droits réservés

Monitoring, Maintenance et Orchestration

Le monitoring de conteneurs Docker implique la surveillance de plusieurs paramètres :

1. **Utilisation des ressources** : Suivi de l'utilisation de la CPU, de la mémoire, du disque, et du réseau par chaque conteneur.
2. **Santé et performances** : Analyse des temps de réponse et de l'activité des processus au sein des conteneurs.
3. **Logs** : Collecte et analyse des logs générés par les conteneurs pour détecter des anomalies ou des erreurs.
4. **Événements** : Surveillance des événements importants tels que les démarrages, les arrêts, et les redémarrages de conteneurs.

- **Docker Stats et Docker Events** : Commandes natives Docker qui fournissent des statistiques de base et des événements de conteneurs en temps réel.
- **Prometheus** : Un système de monitoring et d'alerte open-source qui peut être configuré pour collecter des métriques via l'exposition de points de terminaison par les conteneurs.
- **Grafana** : Utilisé en conjonction avec Prometheus pour visualiser les données collectées sous forme de tableaux de bord dynamiques.
- **cAdvisor** : Un outil de Google qui analyse et expose les performances et les ressources des conteneurs.

Utopios® Tous droits réservés

Mise en Œuvre de Docker Compose

1. Notion de Microservices

Les microservices sont une approche architecturale de développement de logiciels où une application est divisée en de petits services indépendants qui communiquent entre eux via des API bien définies. Chaque microservice est responsable d'une fonctionnalité spécifique de l'application et peut être développé, déployé, et mis à jour indépendamment des autres services.

- **Modularité** : Facilite la maintenance et le développement par des équipes qui peuvent travailler indépendamment les unes des autres.
- **Scalabilité** : Les services peuvent être dimensionnés indépendamment, permettant une meilleure gestion des ressources selon les besoins.
- **Déploiement flexible** : Les microservices peuvent être déployés sur diverses plateformes et technologies sans affecter les autres services.
- **Complexité de gestion** : La gestion de multiples services peut devenir complexe, surtout en termes de réseau, de surveillance, et de sécurité.
- **Latence** : La communication entre services peut introduire une latence supplémentaire.
- **Consistance des données** : Maintenir la consistance des données à travers les services peut être difficile.

Utopios® Tous droits réservés

Mise en Œuvre de Docker Compose

2. Introduction aux architectures N-tiers

Une architecture N-tiers est un modèle de conception logicielle qui sépare une application en plusieurs niveaux ou couches, chacun ayant un rôle spécifique. Typiquement, cela inclut une couche de présentation (front-end), une couche de logique métier (back-end), et une couche de données.

- **Présentation** : Interface utilisateur, gestion des requêtes HTTP, présentation des données.
- **Logique métier** : Traitement des données, règles métier, calculs.
- **Accès aux données** : Communication avec les bases de données, fichiers, ou autres services de stockage.
- **Maintenabilité** : Les modifications dans une couche spécifique n'affectent pas les autres.
- **Scalabilité** : Chaque couche peut être dimensionnée indépendamment selon les besoins.
- **Flexibilité** : Facilite l'intégration de nouvelles technologies et l'évolution de l'application.

Utopios® Tous droits réservés

Mise en Œuvre de Docker Compose

3. Utilisation de Docker Compose

Docker Compose est un outil pour définir et gérer des applications multi-conteneurs avec Docker. Il utilise un fichier YAML pour configurer les services de l'application, les réseaux et les volumes, et permet de gérer l'ensemble avec des commandes simples.

- **Définition de service** : Permet de configurer les propriétés des conteneurs comme l'image à utiliser, les ports à ouvrir, les volumes à monter, etc.
- **Orchestration** : Gère le démarrage et l'arrêt ordonné des conteneurs basé sur les dépendances définies.
- **Isolation** : Chaque service peut être isolé dans son propre conteneur, ce qui améliore la sécurité et réduit les conflits entre services.

Docker Compose est particulièrement utile dans les environnements de développement et de test, où plusieurs conteneurs doivent être lancés simultanément pour simuler une application complète.

- **Configuration centralisée** : Tout est défini dans un fichier `docker-compose.yml`, rendant la configuration transparente et facile à suivre.
- **Gestion de cycle de vie simplifiée** : Permet de démarrer, arrêter, reconstruire et redimensionner les services avec des commandes simples.
- **Isolation** : Chaque ensemble de conteneurs peut fonctionner dans un réseau isolé, ce qui minimise les interférences entre les différents projets ou environnements.

Utopios® Tous droits réservés

Mise en Œuvre de Docker Compose

- **Fichier de configuration**

Le fichier `docker-compose.yml` est le cœur de Docker Compose.

- **version** : Spécifie la version de la syntaxe Docker Compose utilisée.
- **services** : Définit les conteneurs et leurs configurations spécifiques.
- **volumes** : Déclare les volumes pour la persistance des données.
- **networks** : Configure les réseaux pour la communication entre les conteneurs.

```yaml
version: '3.8'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html
    depends_on:
      - db
  db:
    image: postgres:latest
    environment:
      POSTGRES_DB: exemplesdb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
volumes:
  db-data:
networks:
  app-network:
```

Commandes de Base

- **Démarrage des services** : `docker-compose up`
- **Arrêt des services** : `docker-compose down`
- **Visualisation des logs** : `docker-compose logs`
- **Exécution d'une commande dans un service** : `docker-compose exec [service] [command]`

Utopios® Tous droits réservés

Mise en Œuvre de Docker Compose

1. Gestion des Environnements

Docker Compose permet de gérer différents environnements (développement, test, production) en utilisant des fichiers de configuration séparés ou des variables d'environnement. Par exemple, vous pouvez avoir un `docker-compose.yml` pour la base et `docker-compose.override.yml` pour des spécifications locales de développement.

2. Extension et Réutilisation

Les configurations de Docker Compose peuvent être étendues ou réutilisées à travers plusieurs projets grâce à l'héritage de configurations. Vous pouvez utiliser `extends` pour hériter des services définis dans d'autres fichiers Compose, facilitant la réutilisation et la maintenance.

3. Variables d'Environnement

Pour une configuration dynamique, Docker Compose supporte l'utilisation de variables d'environnement dans le fichier `docker-compose.yml`. Ces variables peuvent être définies dans l'environnement d'exécution ou dans des fichiers `.env` spécifiques.

4. Dépendances de Services

Docker Compose permet de définir les dépendances entre services via `depends_on`, ce qui assure que les services dépendants ne démarreront qu'après que les services dont ils dépendent soient opérationnels.

5. Santé des Conteneurs

Avec la propriété `healthcheck`, Docker Compose peut vérifier la santé des conteneurs en exécutant des commandes définies par l'utilisateur pour s'assurer que les services fonctionnent correctement avant de permettre aux autres services de démarrer ou de continuer leur fonctionnement.

Utopios® Tous droits réservés

Mise en Œuvre de Docker Compose

6. Gestion de Réseaux et Volumes

Les réseaux et volumes peuvent être définis et gérés explicitement dans Docker Compose, permettant une isolation et une persistance des données adaptées aux besoins de l'application.

7. Scaling des Services

Docker Compose supporte le scaling de services avec la commande `docker-compose up --scale`. Cela permet d'augmenter ou de diminuer le nombre d'instances d'un service dynamiquement, en fonction des besoins.

8. Intégration avec Docker Swarm

Docker Compose peut être utilisé en tandem avec Docker Swarm pour gérer des déploiements à l'échelle d'un cluster. Le même fichier `docker-compose.yml` peut être utilisé pour déployer des services sur un cluster Swarm, tirant parti de la gestion d'orchestration de Swarm pour la haute disponibilité et la scalabilité.

Utopios® Tous droits réservés

Merci pour votre attention

Des questions ?

Utopios® Tous droits réservés