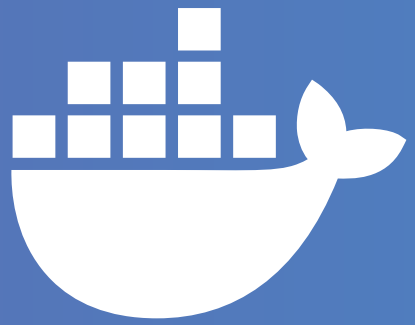
TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE



Docker

Un peu d'histoire

Un peu d'histoire

La plupart des applications fonctionnent sur des serveurs. Autrefois, il n'était possible de faire tourner qu'une seule application par serveur. Windows et Linux ne disposaient tout simplement pas de la technologie nécessaire pour exécuter plusieurs applications sur le même serveur de manière sécurisée.

Un peu d'histoire

Cela se traduisait par une situation complexe : chaque fois qu'une entreprise avait besoin d'une nouvelle application, le département informatique devait acheter un nouveau serveur. Ne connaissant souvent pas les performances requises pour cette nouvelle application, ils optaient pour le serveur le plus gros et le plus coûteux disponible. Cela entraînait l'achat de serveurs surdimensionnés et une perte significative de capital et de ressources.

La création des machines virtuelles

Au milieu de tout cela, **VMware INC** inventa la *machine virtuelle*. Nous disposions désormais de la technologie qui nous permettait d'exécuter en toute sécurité plusieurs applications sur un seul serveur.

Mais cela a créé un nouveau problème... Le fait que chaque VM nécessite son propre OS est un défaut majeur. Chaque OS consomme de la mémoire vive et d'autres ressources qui pourraient être utilisées pour faire fonctionner d'autres applications.

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Conteneurs d'applications

Conteneurs d'applications

Les conteneurs sont une technologie clé dans le paysage du cloud natif. Ils sont utilisés pour emballer et **isoler** les applications avec leurs **dépendances** entières, y compris le système d'exploitation, les bibliothèques système, les scripts, etc.

Cela permet d'assurer que l'application fonctionne de manière cohérente et fiable **dans n'importe quel environnement**, que ce soit en développement, en test ou en production.

Conteneurs d'applications

Un conteneur est **plus léger** qu'une machine virtuelle traditionnelle car il partage le système d'exploitation de l'hôte et n'a pas besoin de son propre système d'exploitation.

Cela rend les conteneurs très efficaces en termes d'utilisation des ressources système et de temps de démarrage.

Conteneurs d'applications

Évolutivité : Les conteneurs peuvent être démarrés et arrêtés rapidement, ce qui facilite leur mise à l'échelle en fonction de la demande. Si la demande pour une application augmente, plus de conteneurs peuvent être lancés pour gérer cette demande.

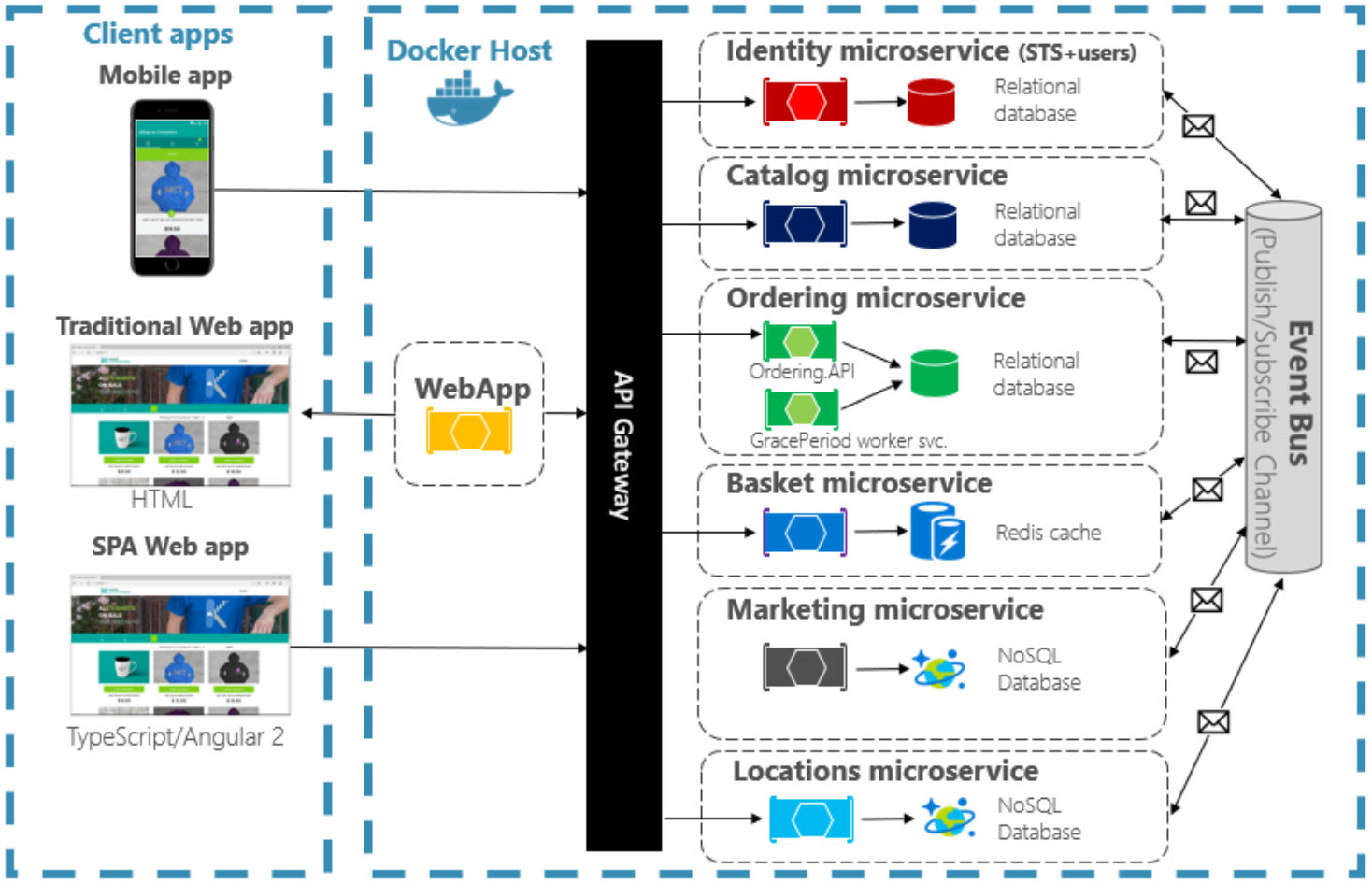
Déploiement continu et livraison continue (CI/CD) : Les conteneurs sont parfaits pour les pipelines CI/CD car ils permettent de déployer rapidement de nouvelles versions d'une application.

Conteneurs d'applications

Isolation : Chaque conteneur fonctionne de manière isolée. Cela signifie qu'un conteneur n'a pas d'effet sur les autres conteneurs et peut avoir ses propres configurations système et logicielle.

Portabilité : Les conteneurs garantissent que l'application fonctionne de manière identique dans tous les environnements. Cela permet de déplacer facilement les applications d'un système à un autre, ou d'un cloud à un autre.

Conteneurs d'applications



UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Docker

Qu'est-ce que Docker

Docker est une plateforme open source de conteneurisation qui simplifie le processus de développement, de distribution et de déploiement d'applications. Les conteneurs Docker encapsulent une application et ses dépendances, garantissant une exécution cohérente quel que soit l'environnement.

Docker répond à la grande problématique du *'ça fonctionne chez moi'*

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Installation de Docker

prérequis

Avant d'installer Docker sur votre machine, assurez-vous de satisfaire les conditions préalables suivantes :

- **Windows** : Docker desktop nécessite Windows 10 Pro/Entreprise ou windows server 2016. Assurez-vous que la virtualisation est activée dans le BIOS.

Installation

1. Téléchargez Docker Desktop à partir du site officiel et suivez les instructions d'installation.
2. lancer la commande :

```
docker run hello-world
```

Celle-ci nous confirme la bonne installation de Docker et nous montre les différentes étapes réalisées suite à la commande docker run.

Les images

Qu'est-ce qu'une image Docker ?

Une image Docker est un modèle léger qui contient tout le nécessaire pour exécuter une application, y compris le code, les dépendances, les variables d'environnement et la configuration. Les images sont créées à partir de Dockerfiles.

Commandes de base

- Affiche la liste des images disponibles sur la machine :

```
docker images
```

- Télécharge une image depuis un registre Docker (par défaut, Docker Hub)

```
docker pull nom_image[:tag]
```

- Crée une image à partir d'un Dockerfile dans le répertoire courant :

```
docker build -t nom_image[:tag]
```

Commandes de base

- Supprime une ou plusieurs images

```
docker rmi nom_image[:tag]
```

La sécurité liée aux images docker

1. Ne **jamais** utilisé `:latest` en production
   - le tag `latest` est ambigu et peux changer à tout moment.
   - Privilégier un tag de version explicite : `ubuntu:20.04`.
2. Vérifier les vulnérabilités **connues** :
   - utilisez des outils comme `docker scan` (**plus disponible dans les versions recente de docker**)
   - ou bien `trivy` : `trivy image ubuntu`

La sécurité liée aux images docker

3. Privilégier les images officielles ou vérifiées
   - Sur DockerHub, repérer les **badges** officiels ou les images **reconnues**.
   - Éviter les images d'utilisateur **inconnus** sans description ou **activité récente**.

La sécurité liée aux images docker

4. Utiliser des **images minimales**
   Exemples :
   - `alpine` (5 Mo environ)
   - Images **distroless** (sans **shell** ni **package** manager)

- **Avantage** :
  - Moins de surface d'attaque
  - Moins de dépendances inutiles

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

DockerFile

Qu'est-ce qu'un Dockerfile ?

Un Dockerfile est un fichier texte qui contient une série d'instructions permettant de créer une image Docker. Il spécifie les dépendances, la configuration et les étapes nécessaires pour exécuter une application.

Structure de base d'un dockerfile

- Un dockerfile suit une structure simple :

```
# Sélection de l'image de base
FROM nom_image[:tag]
# Définition du répertoire de travail
WORKDIR /chemin/du/repertoire
# Copie des fichiers locaux vers l'image
COPY fichier_source destination
# Exécution de commandes dans l'image
RUN commande
# Exposition des ports
EXPOSE port
#Définition de la commande par défaut pour le conteneur
CMD ["commande", "argument"]
```

Instructions principales d'un dockerfile

- **FROM**: Définit l'image de base à partir de laquelle construire.
- **WORKDIR**: Définit le répertoire de travail à l'intérieur de l'image.
- **COPY**: Copie des fichiers locaux vers l'image.
- **RUN**: Exécute des commandes dans l'image pendant la construction.
- **EXPOSE**: Spécifie les ports sur lesquels le conteneur écoute.
- **ENV**: Définit des variables d'environnement.
- **CMD**: Définit la commande par défaut pour le conteneur.

Démonstration pratique

- **Création d'une image Node.js :**

```
FROM node:14
WORKDIR /app
COPY package*.json
RUN npm install
COPY ..
EXPOSE 3000
CMD ["npm", "start"]
```

Démonstration pratique

- **Construction de l'image :**

```
docker build -t mon-app-nodejs:1.0
```

- **Exécution d'un conteneur avec l'image créée :**

```
docker run -p 3000:3000 mon-app-nodejs:1.0
```

- Ouvrez votre navigateur à http://localhost:3000 pour accéder à l'appication Node.js.

- **Suppression de l'image (facultatif) :**

```
docker rmi mon-app-nodejs:1.0
```

Gestion des tags d'images

- Les tags permettent de versionner les images. Par défaut, le tag est `latest`
- **Ajout d'un tag lors de la construction**

```
docker build -t mon-app-nodejs:version
```

- **Modification du tag d'une image existante :**

```
docker tag mon-app-nodejs:version mon-app-nodejs:nouvelle_version
```

La sécurité dans les dockerfile

1. Éviter `latest` et utiliser des version fixe

```
#  À éviter
FROM ubuntu:latest

#  Bonnes pratiques
FROM ubuntu:20.04
```

La sécurité dans les dockerfile

2. Ne jamais exécuter en `root`

```
#  Créer un utilisateur non priviliégié
RUN addgroup --system appgroup && adduser --system --ingroup appgroup appuser
USER appuser
```

- /!\ les images `alpine`, `node`, `nginx` proposent parfois déjà un utilisateur.

La sécurité dans les dockerfile

- Ne jamais copier `.env`, `credentials.json` etc.

```
COPY . . # Cela risque de copier des fichiers sensibles

# Spécifier uniquement les fichiers nécessaires
COPY app/ /app/
COPY requirements.txt .
```

- /!\ Vous pouvez ajouter un `.dockerignore`

Exercice 1 — "Dockerfile minimal Python"

- Écrire un `Dockerfile` pour un script Python simple (ex: transformation de fichier CSV → JSON)
- Construire et exécuter

Exercice 2 — "Personnalisation : arguments & entrypoint"

- Améliorer l'image :
  - Ajouter `ARG` ou `ENV`
  - Définir un `ENTRYPOINT`
  - Taguer correctement l'image

Les conteneurs

Qu'est-ce qu'un conteneur Docker

- Un conteneur Docker est une instance exécutable d'une image docker. Il isole l'application et ses dépendances du système hôte, garantissant une portabilité et une cohérence d'exécution.

Commandes de base pour travailler avec des conteneurs Docker

- **Créer et démarrer un conteneur à partir d'une image :**

```
docker run [options] nom_image[:tag] [commande] [arguments]
```

- **Affiche les conteneurs en cours d'exécution :**

```
docker ps
```

- **Exécute une commande à l'interieur d'un conteneur en cours d'exécution :**

```
docker exec [options] nom_conteneur commande [arguments]
```

Commandes de base pour travailler avec des conteneurs Docker

- **Arrête un ou plusieurs conteneurs :**

```
docker stop nom_conteneur
```

- **Supprime un ou plusieurs conteneurs :**

```
docker rm nom_conteneur
```

- **Affiche les logs d'un conteneur :**

```
docker logs nom_conteneur
```

options -p

- Cette option permet de rediriger tout le trafic envoyé au port choisi sur votre machine vers le port choisi à l'interieur du conteneur.

```
docker run -p port_hote:port_container nom_image[:tag] [commande] [arguments]
```

Démonstration pratique

Scénario : Exécution d'un conteneur MySQL

1. **Téléchargement et exécution d'un conteneur MySQL**:

```
docker run -d --name mon-mysql -e MYSQL_ROOT_PASSWORD=my_secret_pw mysql:latest
```

   - `-d`: Mode détaché.
   - `--name mon-mysql` : Attribution d'un nom au conteneur.
   - `-e MYSQL_ROOT_PASSWORD=my-secret-pw`: Définition d'un mot de passe pour l'utilisateur root.

Scénario : Exécution d'un conteneur MySQL

2. **Vérification de l'exécution du conteneur**:

```
docker ps
```

3. **Connexion à MySQL depuis un client MySQL**

```
docker exec -it mon-mysql mysql -uroot -pmy-secret-pw
```

4. **Arrêt et suppression du conteneur (facultatif)**:

```
docker stop mon-mysql
docker rm mon-mysql
```

Scénario: Exécution d'un conteneur serveur web avec caddy

1. **Téléchargement et exécution d'un conteneur caddy**:

```
docker run -p 8080:80 --name caddy-container -d caddy
```

- `-d`: Mode détaché
- `--name cady-container` : Attribution d'un nom au conteneur.
- `-p 8080:80`: Cette option permet de rediriger tout trafic envoyé au port 8080 sur votre machine vers le port 80 à l'interieur du conteneur.

Scénario: Exécution d'un conteneur serveur dotnet

1. **Téléchargement et exécution d'un conteneur dotnet**:

```
docker run -it -p 5000:5000 -p 5001:5001 mcr.microsoft.com/dotnet/sdk:latest
```

- `-p 5000:5000 -p 5001:5001`: Cette option permet de rediriger tout trafic envoyé au port 5000 et 5001 sur votre machine vers le port 5000 et 5001 à l'interieur du conteneur.

La sécurité des conteneurs

1. Éviter les conteneurs avec **privilèges** élevés :
   - Ne **JAMAIS** exécuter un conteneur avec `--privileged` sauf nécessité absolue.
2. Utiliser un utilisateur **non-root** :
   - Par défaut, de nombreux conteneurs tournent en tant que `root`. Créer un utilisateur **dédié** et l'utiliser dans le **dockerfile** ou via `--user`

La sécurité des conteneurs

3. **Limiter** les **capacitées** du conteneur :

- L'option `--cap-drop` permet de **supprimer des capacitées du conteneur**. C'est capacitées sont des **privilèges spécifiques** que possède un processus, indépendamment du fait qu'il s'éxecute en **root** ou non.

```
docker run --cap-drop=<capacité> image
```

- peut être utilisée plusieurs fois.

Exemples

- Supprimer **toutes** les capacitées :

```
docker run --cap-drop=ALL ubuntu
```

- Supprimer **NET_RAW** (empêche ping, etc.) :

```
docker run --cap-drop=NET_RAW ubuntu
```

- `--cap-drop` permet de retirer une ou plusieurs de ces capacités pour réduire la **surface d'attaque**.

Liste d’exemples de capacités Docker importantes :

Capacité	Description
`NET_ADMIN`	Modifier les interfaces réseau, activer IP forwarding
`SYS_ADMIN`	Capacité très puissante (quasi root)
`CHOWN`	Changer le propriétaire d’un fichier
`SETUID`	Changer d'identifiant utilisateur
`MKNOD`	Créer des fichiers spéciaux
`SYS_TIME`	Changer l'heure système
`NET_RAW`	Envoyer des paquets bruts

Exercice 1 — "Hello Container"

- Objectif : comprendre image → container
- Lancer un container **Alpine** et exécuter un `echo "Hello from container"`

Exercice 2 — "Exploration d'image officielle"

- Lancer un container **nginx** et observer :
  - Processus actif (`docker exec`)
  - Fichiers présents (`docker exec ls`)
  - Logs (`docker logs`)

Exercice 3 — "Sécuriser le runtime"

- Lancer les containers avec :
  - `--cap-drop=ALL`
  - `--read-only`

Gestion des réseaux Docker

- **Créer un réseau Docker**:

```
docker network create nom_reseau
```

- **Connecter un conteneur à un réseau**:

```
docker network connect nom_reseau nom_conteneur
```

- **Afficher la liste des réseaux Docker**:

```
docker network ls
```

Les volumes

Qu'est-ce qu'un volume Docker ?

- Un volume Docker est un mécanisme de stockage persistant qui existe en dehors du cycle de vie du conteneur. Il permet le partage de données entre les conteneurs, ainsi que la persistance des données même si le conteneur est supprimé.

Commandes de base pour travailler avec des volumes Docker.

- **Créer un volume Docker**:

```
docker volume create nom_volume
```

- **Afficher la liste des volumes Docker**

```
docker volume ls
```

- **Monter un volume dans un conteneur pendant son démarrage.**

```
docker run -v nom_volume:/chemin/du/volume nom_image
```

Commandes de base pour travailler avec des volumes Docker.

- **Afficher des informations détaillées sur un volume**

```
docker inspect nom_volume
```

- **Supprime un conteneur et le volume monté avec l'option `-v`**

```
docker rm -v nom_conteneur
```

- **Supprime un ou plusieurs volumes Docker**

```
docker volume rm nom_conteneur
```

Démonstration Pratique

Scénario: utilisation d'un volum pour persister des données MySQL

1. Création d'un volume MySQL

```
docker volume create mysql_data
```

2. Exécution d'un conteneur MySQL avec le volume monté

```
docker run -d --name mon-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -v mysql_data:/var/lib/mysql mysql:latest
```

Scénario: utilisation d'un volume pour persister des données MySQL

3. Vérification de l'exécution et des volumes montés

```
docker ps
docker inspect mon-mysql
```

4. Arrêt et suprpession du conteneur (facultatif)

```
docker stop mon-mysql
docker rm mon-mysql
```

- Le volume `mysql-data` persiste même si le conteneur est supprimé

Gestion des types de volumes

- Montage d'un répertoire du système hôte dans un conteneur.

```
docker run -v /chemin/local:/chemin/conteneur nom_image
```

- Création explicite d'un volume nommé.

```
docker volume create nom_volume
```

- Création automatique d'un volume sans spécifier de nom.

```
docker run -v /chemin/conteneur nom_image
```

La sécurité des volumes

- Monter les volumes en lecture seule quand c'est possible.

```
-v /data/config:/app/config:ro
```

- ou en ligne de commande avec l'option `--read-only`

Exercice 1 — "Communication container : service BDD"

- Lancer un container mysql avec un volume persistant
- Écrire un script Python qui insère des données en se connectant à la BDD
- Relier les containers via un `docker network` personnalisé

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Docker compose

Qu'est ce que docker compose ?

- Docker compose est un outil qui permet de définir et de gérer des applications multi-conteneurs Docker. Il utilise un fichier YAML pour configurer les services, les réseaux, les volumes, et les configurations nécessaires à l'application.

Commandes de base pour travailler avec docker compose

- **Démarrer les conteneurs selon la configuration du fichier docker-compose.yml**

```
docker-compose up
```

- **Arrête ou supprime les conteneurs, les réseaux et les volumes définis dans le fichier**

```
docker-compose down
```

- **affiche l'état des services**

```
docker-compose ps
```

Commandes de base pour travailler avec docker compose

- **Affiche les logs des services**

```
docker-compose logs nom_service
```

- **Exécute une commande dans un service**

```
docker-compose exec nom_service commande
```

Démonstration pratique

- **Scénario: déploiement d'une application web**

1. **Création du fichier docker-compose.yml**
2. **Démarrage de l'application**

```
docker-compose up
```

3. **Accès à l'application web et vérification de l'état**
   - Ouvrez votre navigateur à http://localhost:8080
   - L'application node est accessible à http://localhost:3000

La sécurité dans un docker compose

- Il suffit de mettre en place toutes les pratiques vues précédemment.
- quelques exemples :
  - Monter les volumes de façon sécurisée

```
volumes:
  - type: volume
    source: app_data
    target: /app
    read_only: true
```

quelques exemples :

- Utiliser des réseaux isolés par service :

```
networks:
  frontend:
  backend:

services:
  web:
    networks: [frontend]
  db:
    networks: [backend]
```

quelques exemples :

- Lancer les conteneurs avec le strict minimum de privilèges :

```
services:
  app:
    image: myapp
    user: "1001:1001" # évite root
    read_only: true
    cap_drop:
      - ALL
```

Merci pour votre attention

Des questions ?