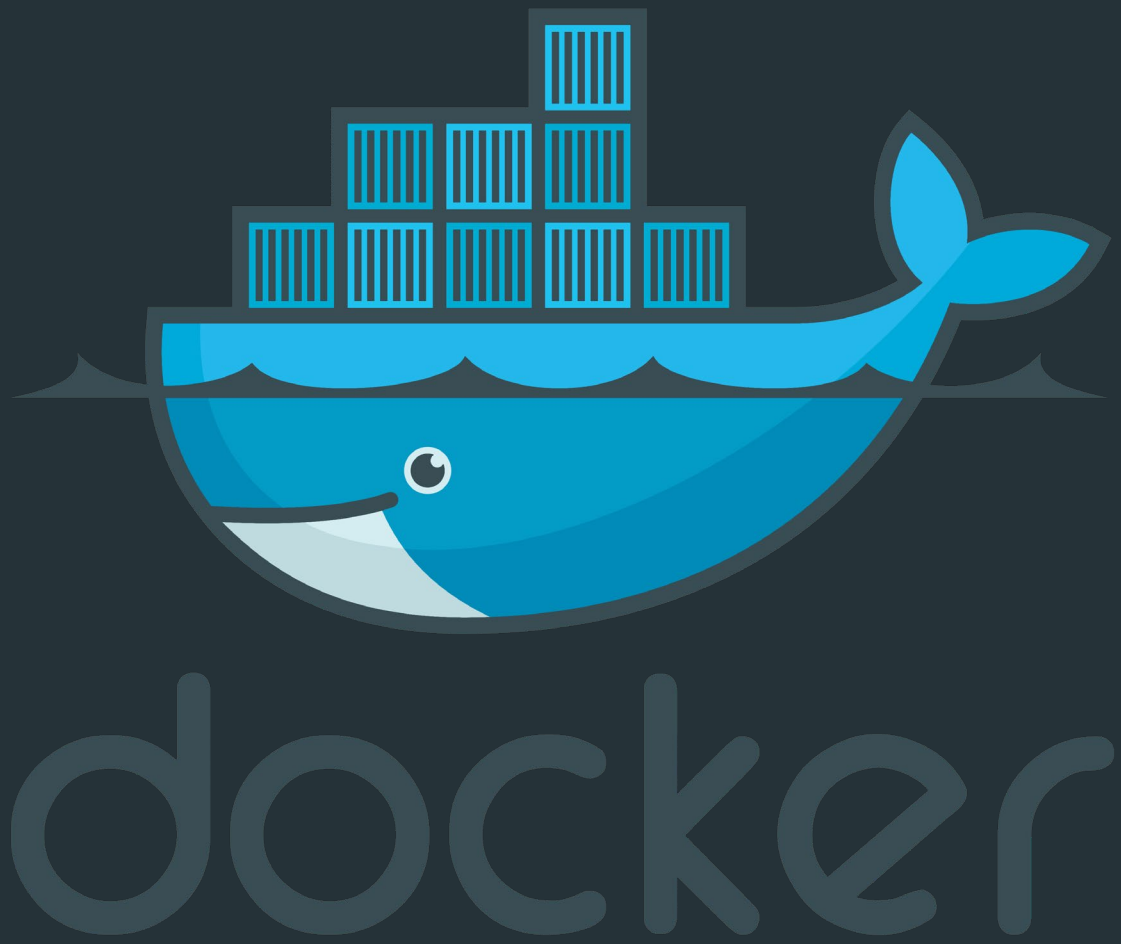
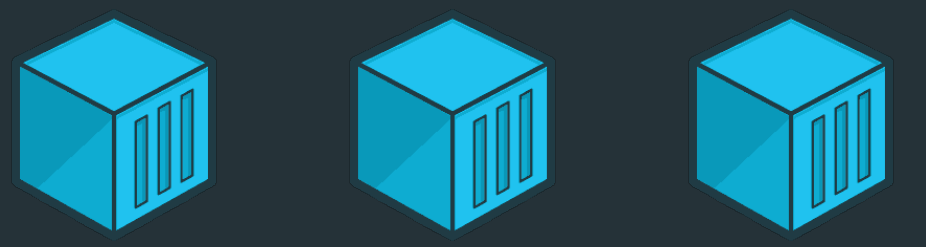


Sam Vie
Maxime Tueux

Pourquoi utiliser Docker ?

- Meilleur portabilité des applications
- Environnement isolé et sécurisé
- Légèreté et rapidité des conteneurs face aux machines virtuelles
- Utilisation efficace des ressources système
- Facilité de partage via des registres d’images
- Accélération des cycles de livraison en production

Qu’est ce qu’un container ?



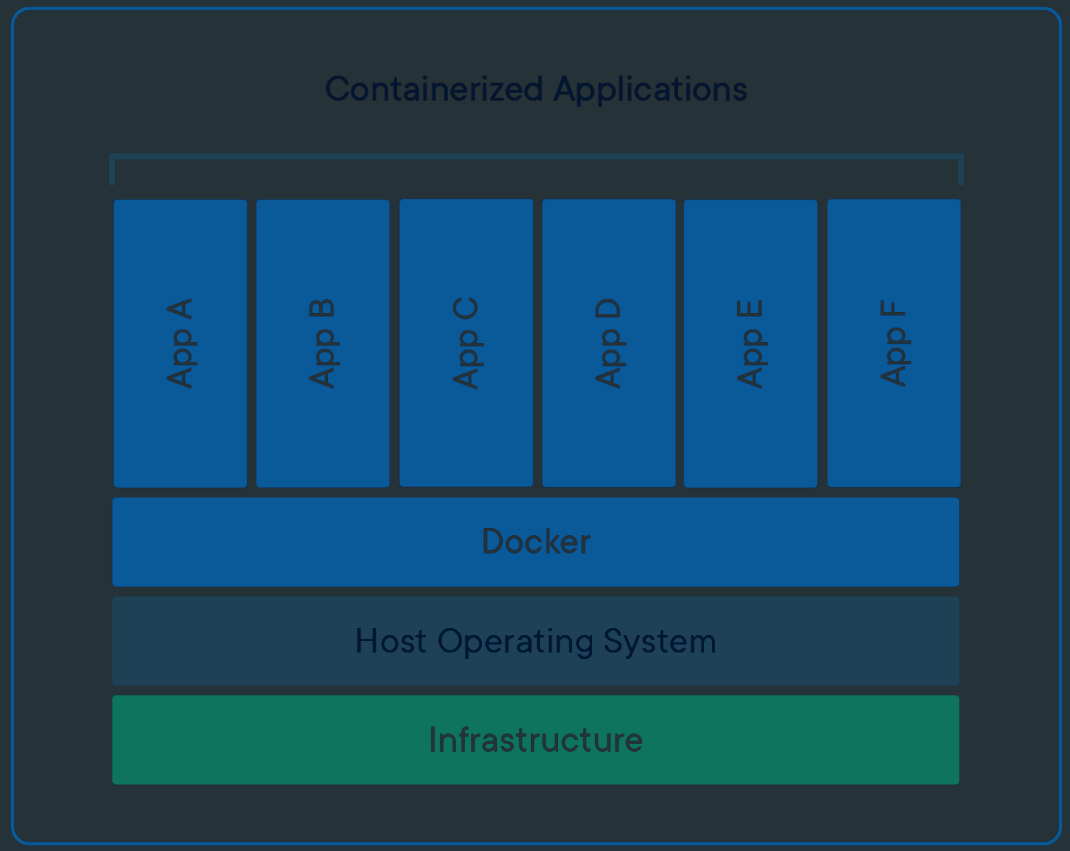
- Processus isolés
- Trafic réseau isolé
- Système de fichiers isolé

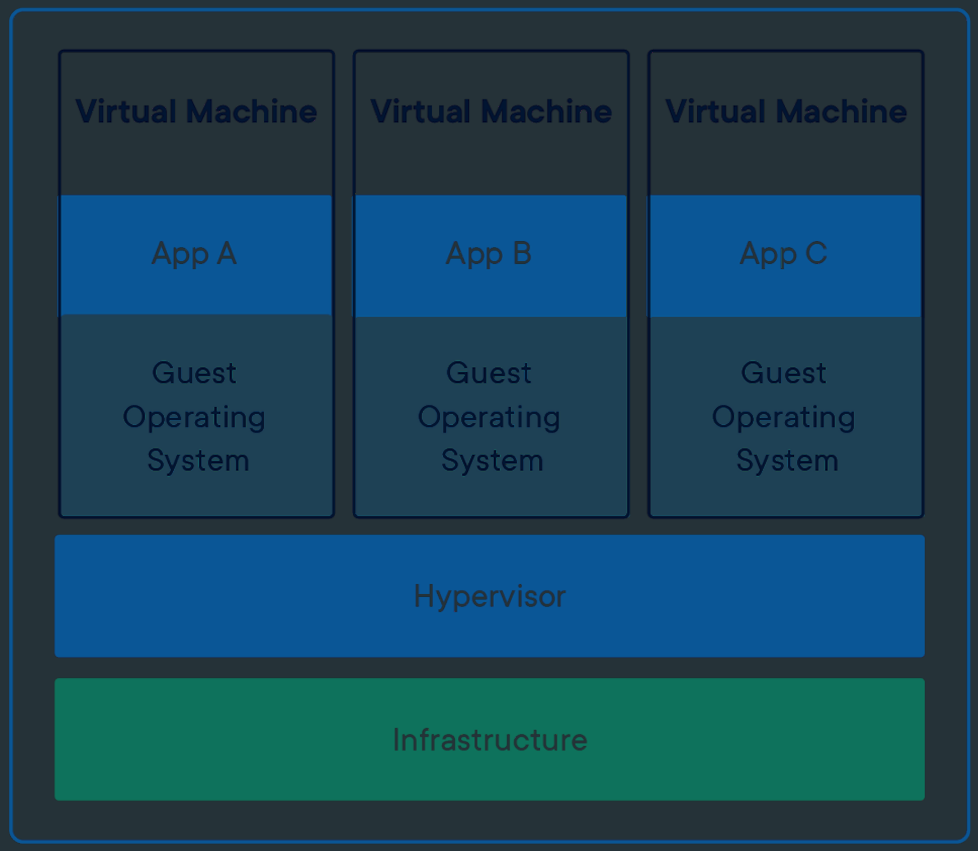
Docker

Système d’exploitation hôte

Infrastructure

Conteneurs et machines virtuelles





Source: https://www.docker.com

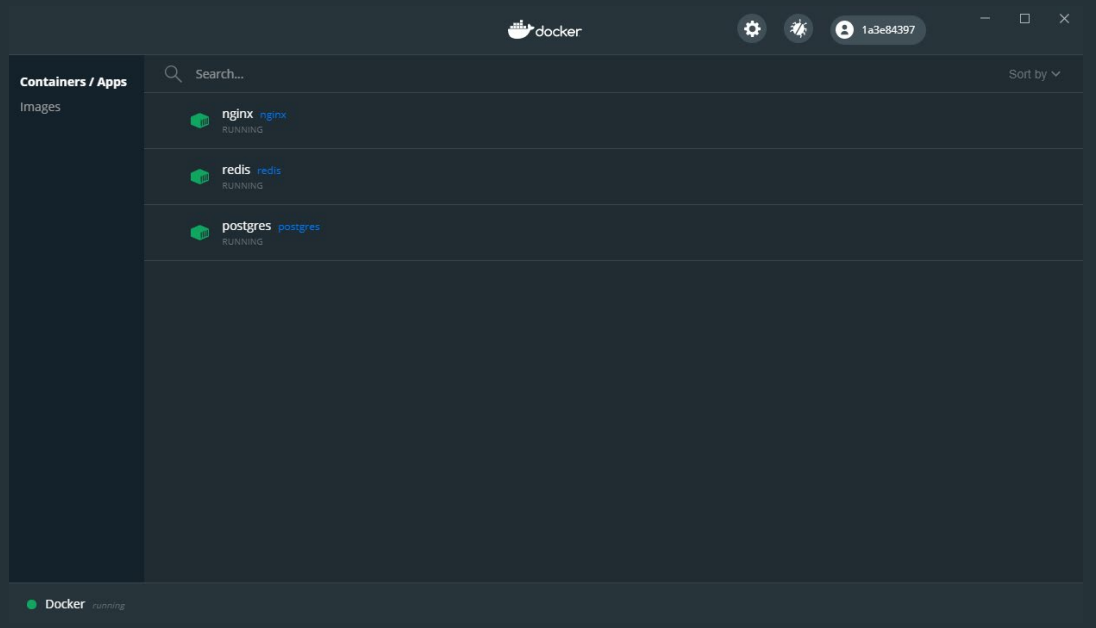
Conteneurisation et virtualisation ne sont pas antagonistes mais complémentaires

Docker sur Windows

WSL 1 / WSL 2

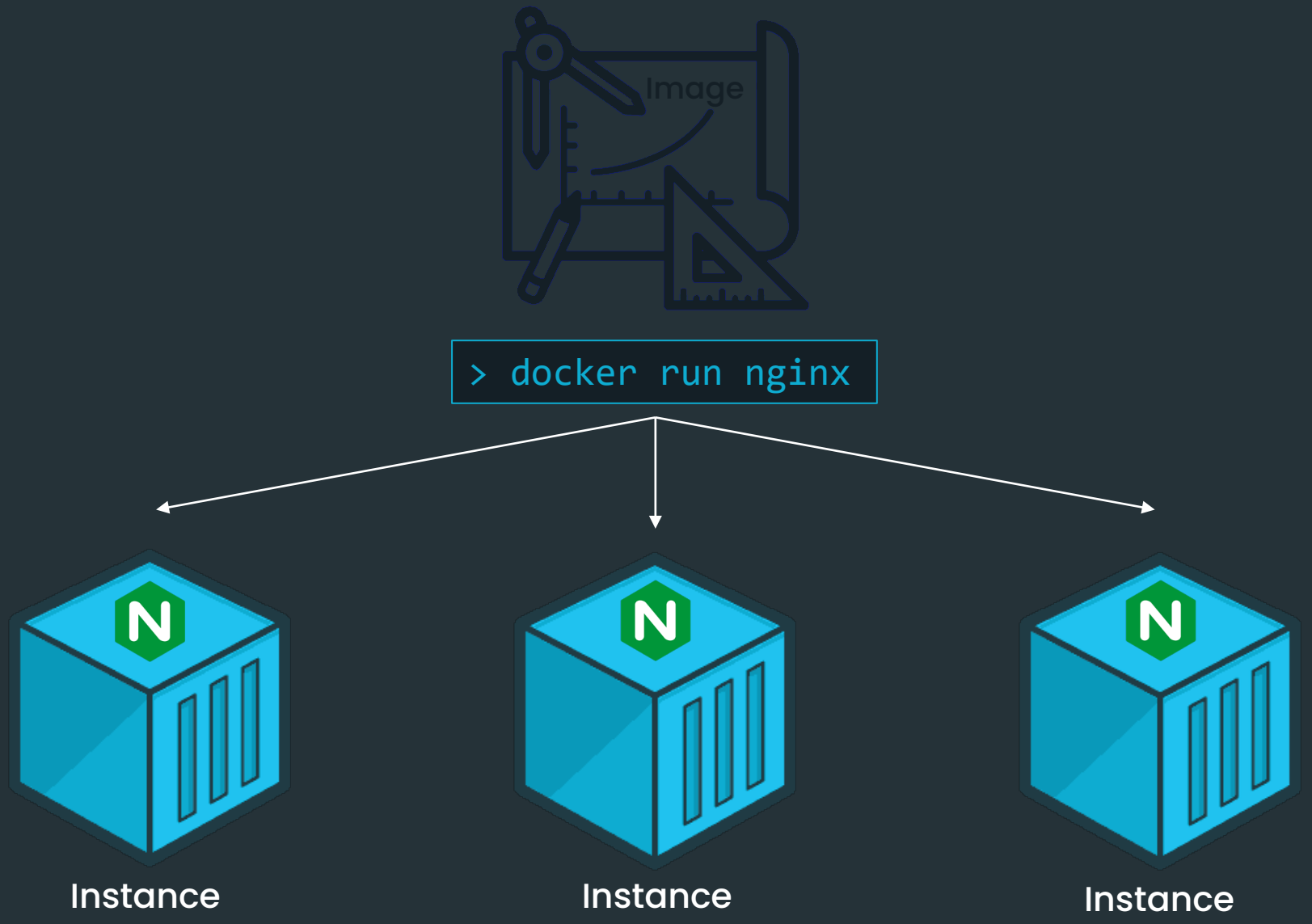
Windows

Infrastructure



Docker Desktop

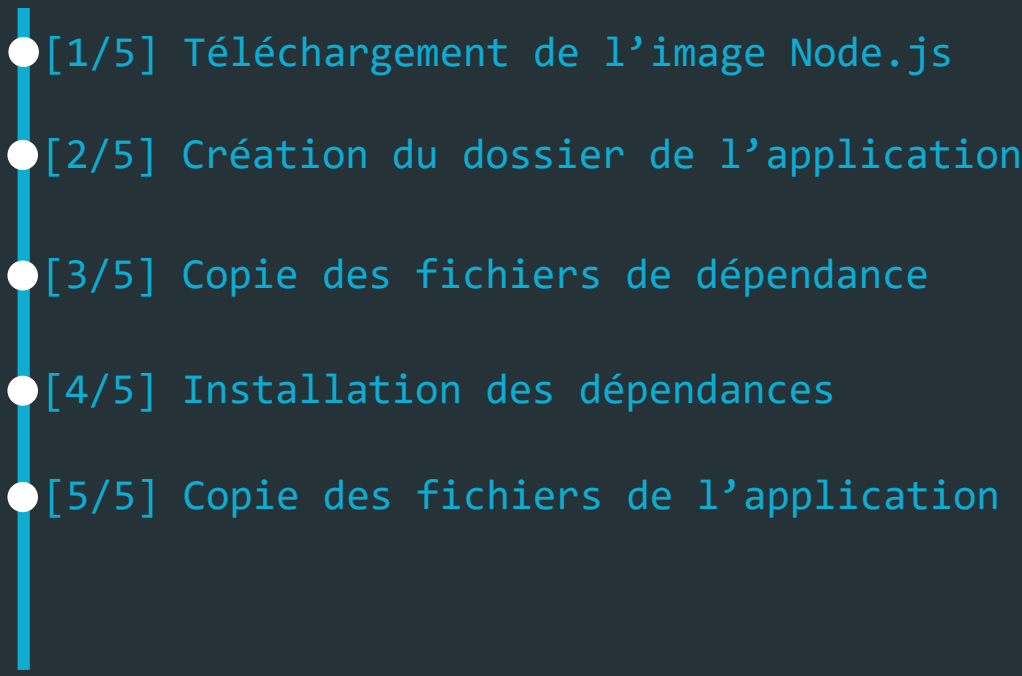
Images Docker



Dockerfile

```
FROM node:14

WORKDIR /usr/src/app

COPY package*.json

RUN npm install

COPY . .

CMD ["node", "server.js"]
```

Architecture en couche avec système de cache



Afin de profiter du cache des couches, celles-ci doivent être organisées intelligemment

Multi-stage builds

```
# Building Stage
FROM node as builder
WORKDIR /usr/src/app
COPY package* .
COPY src/ ./src/
RUN ["npm", "install"]

# Testing Stage
FROM node as unit-tests
WORKDIR /usr/src/app
COPY --from=builder /usr/src/app .
RUN ["npm", "test"]

# Execution Stage
FROM node
WORKDIR /usr/src/app
COPY --from=builder /usr/src/app .
COPY --from=builder /usr/src/app/package* .
RUN ["npm", "start"]
```

RUN, CMD, ENTRYPOINT

Ajout d'une nouvelle couche à l'image

```
RUN ["npm", "install"]    # Exec format
```

```
RUN npm install           # Shell format
```

Commande par défaut du container

```
CMD ["nginx"]             # One per file
```

Point d'entrée des arguments

```
ENTRYPOINT ["echo"]       # One per file
```

Convention de nommage des images Docker

- Un tag identifie une version donnée d'une image
- Le tag spécial latest désigne la dernière version publiée
- Plusieurs tags peuvent pointer sur la même image

Builds et tags

```
> docker build -t <tag> [-f <file>] <context>
```

```
> docker tag <image> <new tag>
```

- Par défaut, Docker cherche un fichier nommé « Dockerfile>
- Le <contexte> est le dossier dans lequel le Dockerfile est contenu
- Docker va créé une image en suivant les instructions du Dockerfile et la stocké sous le nom <tag>
- Il est possible de créer et de supprimer des tags pointant sur une image (mécanisme d'indirection)

Journaux et troubleshooting

```
> docker logs <container>        # Print STDOUT and STDERR
```

```
> docker exec -it <container> sh          # Launch a shell
```

- Une application conteneurisée doit envoyer ses logs sur la sortie standard et non un fichier sur le conteneur
- Il est possible de lancer un shell au sein d'un conteneur si l'image inclus un tel programme

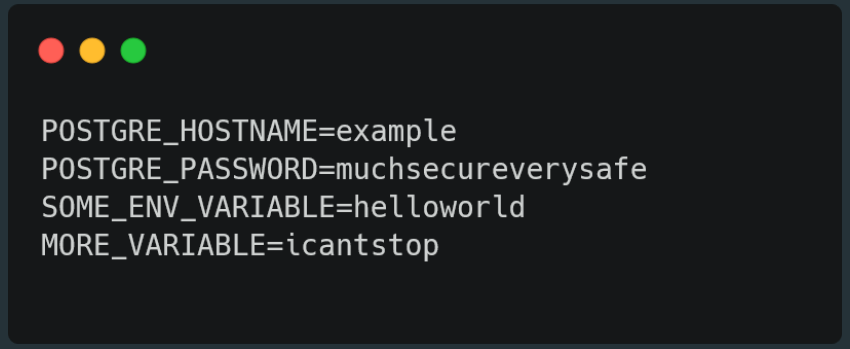
Commandes de base

```
> docker ps                          # List active containers
> docker ps -a                       # List all containers
> docker logs <container>            # Display container's logs
> docker run -d <image>              # Launch a detached container
> docker stop <container>            # Stop a container
> docker rm <container>              # Delete a container
> docker pull <image>:<tag>          # Pull an image on localy
> docker rmi <image>:<tag>           # Delete an image
> docker images                      # List all localy available images
```

Variables d'environnement

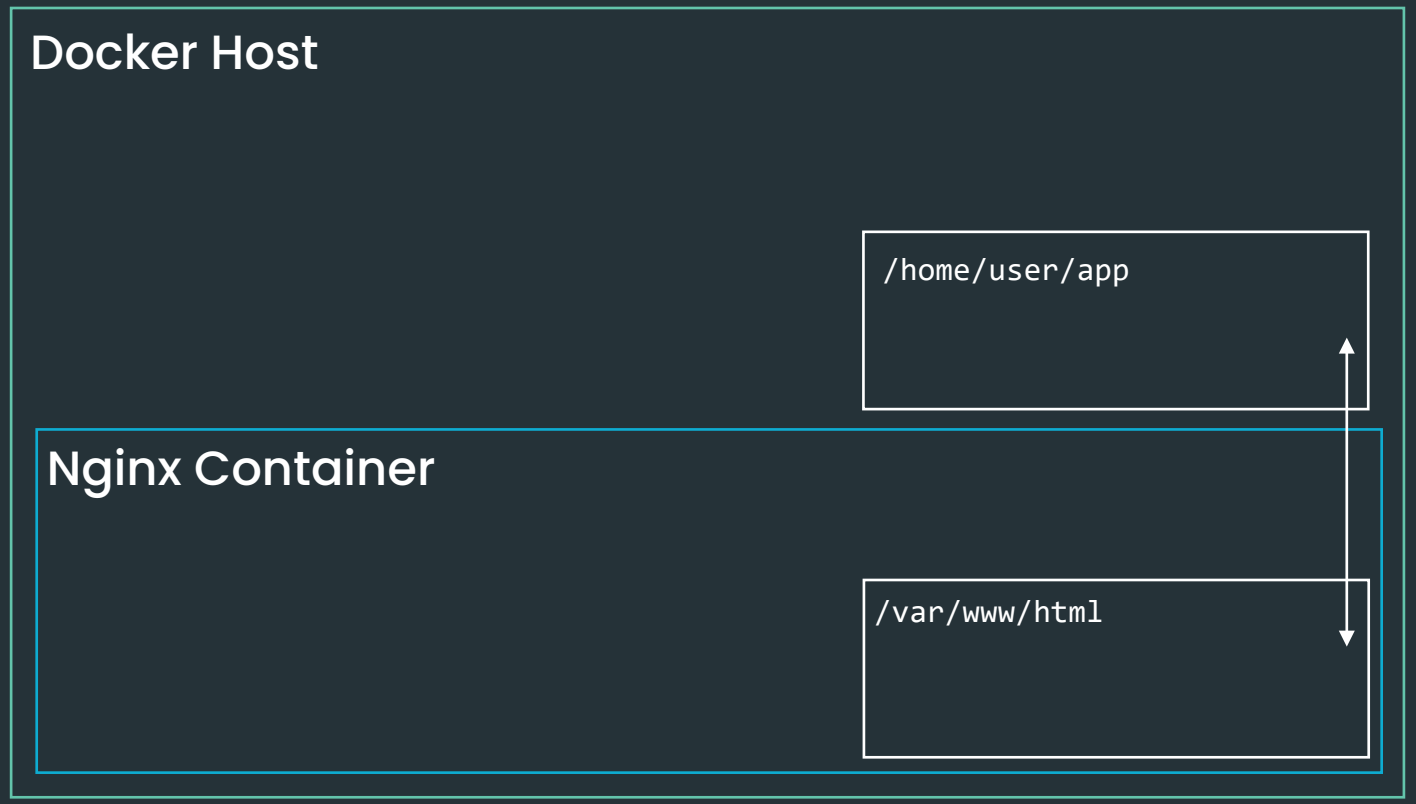
```
> docker run -e POSTGRES_PASSWORD=secret postgres
```

```
> docker run --env-file path/to/file.env postgres
```



Fichier .env stockant des variables d'environnements

Volumes Docker

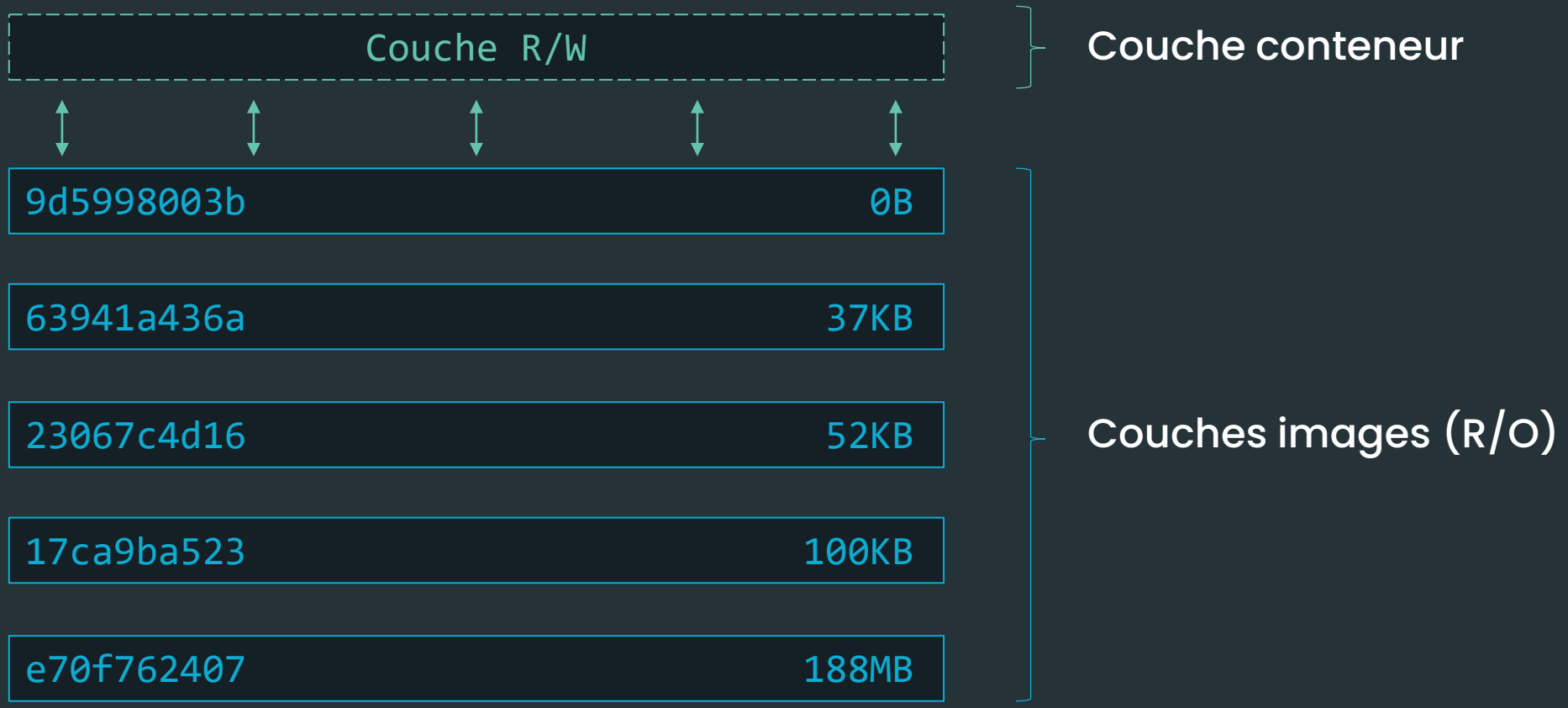



```
> docker run -v /home/user/app:/var/www/html nginx # Bind mount
```

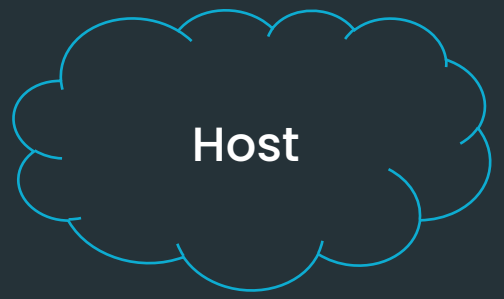
```
> docker run -v mydockervolume:/var/www/html nginx # Volume mount
```

- Les conteneurs sont incapables de faire persister leur état après arrêt sans un stockage externe
- Les volumes permettent d'assurer la persistance des données après redémarrage
- Cette impossibilité à persister les données d'un conteneur est dû au mécanisme de « copy-on-write »

Mécanisme de copy-on-write



Gestion de réseaux



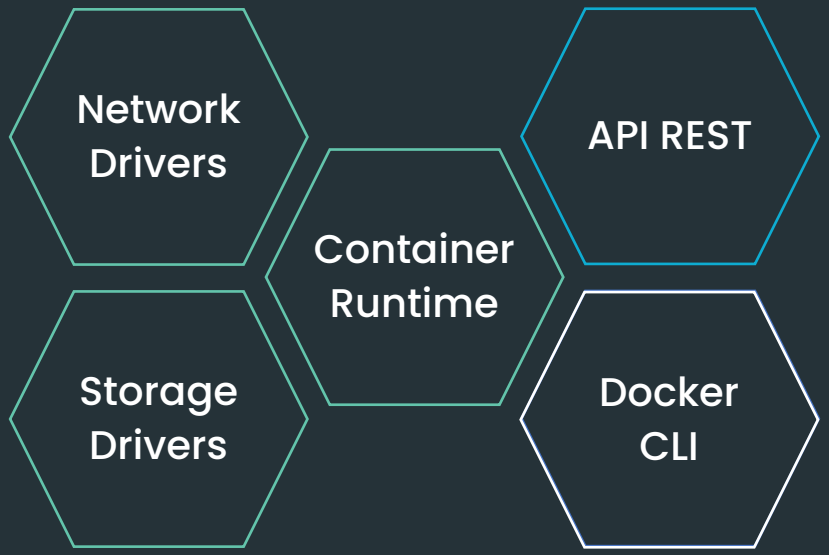

```
> docker run --network=bridge my-app
```

```
> docker run -p 8080:80/tcp nginx
```

```
> docker network create mynetwork \
  --driver=bridge \
  --subnet=192.168.0.0/16
```

Conteneurs et DNS

Anatomie de Docker



- Docker est une agrégation de plusieurs logiciels,
- Le container runtime s'appuie sur divers drivers (stockage, réseau)
- Le runtime est exposé via une API REST
- Le CLI docker communique via cette API

Container Runtime

```
> docker run ubuntu ps aux
```

```
> docker run --cpus=1 --memory=1000m nginx
```

- Comment le conteneur « ubuntu » est-il isolé des autres processus de l'hôte ?
- Comment les limites définies par ressources sont-elles respectées ?

Registres d'images

Docker Hub est le registre par défaut

```
> docker pull nginx
```

```
> docker push organization/app
```

https://hub.docker.com

Utilisation d'un registre distant

```
> docker login acme-registry.com
```

```
> docker pull acme-registry.com/acme/app
```

Sécurité des conteneurs

- Le processus conteneurisé est lancé par un utilisateur adéquat (i.e. pas super-utilisateur sauf exception)
- Ne pas lancer de conteneurs privilégiés (--privileged)
- S'assurer que les capacités Linux définies sont bien nécessaires
- Signer cryptographiquement les images afin de limiter les attaques sur la chaîne de livraison

```
FROM ubuntu
# Création d'un utilisateur classique
RUN groupadd -g 1000 appuser \
    useradd -r -u 1000 -g appuser appuser

# Changement de l'utilisateur (celui par défaut
# étant super-utilisateur)
USER appuser

CMD bash
```

Distroless images

- Images n'embarquant que le minimum pour un type d'application (pas de shell, de librairies, etc.)
- Limites les risques de sécurité et la taille de l'image finale
- Le tag debug embarque un shell dans l'image

```
FROM golang:1.18 as build

WORKDIR /go/src/app
COPY . .

RUN go mod download
RUN CGO_ENABLED=0 go build -o /go/bin/app

# Distroless image
FROM gcr.io/distroless/static-debian11
COPY --from=build /go/bin/app /
CMD ["/app"]
```

https://github.com/GoogleContainerTools/distroless