🐳 Cours sur Docker Networking : Bridge vs Host vs Overlay

📌 1. Introduction à Docker Networking

Quand vous créez un conteneur Docker, celui-ci est connecté par défaut à un réseau virtuel. Docker propose plusieurs **pilotes de réseau** pour différents cas d'usage.

Les principaux pilotes de réseau :

Pilote	Description rapide
`bridge`	Réseau privé par défaut pour les conteneurs sur un même hôte
`host`	Utilise directement le réseau de l'hôte sans isolation
`overlay`	Réseau multi-hôtes pour les clusters Swarm
`none`	Conteneur sans réseau (isolé)

🌉 2. Réseau `bridge` (réseau par défaut)

◇ Fonctionnement :

- Chaque conteneur reçoit une IP locale (ex: `172.17.0.x`).
- Docker crée une interface virtuelle `docker0` sur l'hôte.
- Communication possible entre les conteneurs **via IP ou nom de conteneur**.

☑ Avantages :

- Isolation simple
- Contrôle sur le trafic
- Parfait pour développement et tests

⚠ Inconvénients :

- Pas optimisé pour les performances
- Complexe à exposer sans `-p` ou `-P`

□ Exemple :

```
docker run -dit --name webapp nginx
docker exec -it webapp ping google.com
```

🏠 3. Réseau `host`

◇ Fonctionnement :

- Le conteneur **partage le réseau de l'hôte**.
- Pas d'isolation réseau : le port utilisé par le conteneur doit être libre sur l'hôte.

☑ Avantages :

- Moins de latence
- Accès direct au réseau de l'hôte

⚠ Inconvénients :

- Moins sécurisé
- Conflits de ports possibles

□ Exemple :

```
docker run -dit --network host nginx
```

🌐 4. Réseau `overlay` (multi-hôtes)

◇ Fonctionnement :

- Permet à des conteneurs sur **plusieurs hôtes Docker** de communiquer.
- Nécessite **Docker Swarm ou Kubernetes**.

☑ Avantages :

- Réseau distribué
- Idéal pour microservices

⚠ Inconvénients :

- Configuration avancée
- Besoin d'un cluster Swarm

□ Exemple (Swarm requis) :

```
docker network create -d overlay my_overlay_net
docker service create --name web --network my_overlay_net nginx
```

🛡 5. Sécuriser les conteneurs avec un `bridge` personnalisé

Créer un réseau personnalisé permet :

- D’éviter les conflits avec `docker0`
- D’attribuer des sous-réseaux spécifiques
- D’isoler des groupes de conteneurs

□ Étapes :

☑ Étape 1 : Créer le réseau

```
docker network create \
  --driver bridge \
  --subnet 192.168.100.0/24 \
  secure_net
```

☑ Étape 2 : Lancer les conteneurs dans ce réseau

```
docker run -dit --name app1 --network secure_net nginx
docker run -dit --name app2 --network secure_net alpine
```

☑ Étape 3 : Tester la communication

```
docker exec -it app2 ping app1
```

☑ Étape 4 : Limiter l’accès avec une règle de pare-feu (optionnel)

Utiliser `iptables` ou définir des **politiques réseau personnalisées** via des outils comme **Cilium** ou **Calico** (avancé, surtout en prod ou avec Kubernetes).

📦 Résumé

Type de réseau	Isolation	Performance	Multi-hôte	Utilisation recommandée
`bridge`	☑	Moyenne	❌	Apps simples, local dev
`host`	❌	Haute	❌	Performance critique
`overlay`	☑	Bonne	☑	Cluster, microservices

🛠 Bonus : Bonnes pratiques

- Utilisez des réseaux personnalisés pour **éviter les collisions**.
- Ne jamais exposer des ports non nécessaires.
- Limitez les permissions (utilisez des utilisateurs non-root dans les conteneurs).
- Monitorer les connexions inter-containers avec `docker network inspect`.