Les réseaux docker et Docker compose

Les types de réseaux docker

Le Réseau Bridge (Pont)

Le réseau **Bridge** est le type par défaut et le plus couramment utilisé par Docker. Il agit comme un **routeur virtuel** qui connecte les conteneurs sur le **même hôte Docker**, leur permettant de communiquer entre eux par leur nom ou leur adresse IP interne. C'est le choix idéal pour les applications multi-conteneurs en développement local, car il fournit une bonne isolation réseau tout en nécessitant un **mapping de port** (-p) pour rendre le conteneur accessible depuis l'extérieur de l'hôte.

Le Réseau Host (Hôte)

Le réseau **Host** offre le niveau d'isolation le plus faible en permettant au conteneur de **partager directement la pile réseau** de la machine hôte. Le conteneur utilise l'adresse IP de l'hôte, ce qui élimine les coûts liés à la traduction d'adresse réseau (NAT). Ce type est pertinent lorsque la **performance brute** ou l'accès direct aux ports de l'hôte est critique, mais il est moins sécurisé car il expose le conteneur au réseau de l'hôte.

Le Réseau None (Aucun)

Le réseau **None** isole complètement le conteneur du réseau. Il ne lui attribue **aucune interface réseau** (sauf l'interface *loopback* interne), le rendant incapable de communiquer avec d'autres conteneurs ou avec l'extérieur. Il est spécifiquement utilisé pour les conteneurs dont la seule fonction est d'effectuer des tâches de traitement ou de calcul qui ne nécessitent aucune entrée ou sortie réseau.

Réseaux Bridge Personnalisés

Créer un réseau **Bridge personnalisé** est la **meilleure pratique** en développement pour remplacer le réseau default bridge. L'intérêt principal est de permettre à vos conteneurs de se **résoudre mutuellement par leur nom** (résolution DNS intégrée), simplifiant la configuration. On le crée avec docker network create et on y attache les conteneurs lors de leur lancement, offrant une organisation et une isolation plus propres de vos applications

- Lance un conteneur Nginx en arrière-plan (-d) et télécharge l'image, le connectant automatiquement au réseau bridge par défaut.

```
PS C:\Users\chayma\Desktop\ing3\procedure de test> docker run -d --name web nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
de57a609c9d5: Download complete
108ab8292820: Download complete
53d743880af4: Download complete
05feb73171bf: Downloading [=======================================>           ]  25.17MB/29.97MB
0e4bc2bd6656: Extracting 2 s
77fa2eb06317: Download complete
192e2451f875: Download complete
```

- Affiche les détails de configuration du réseau bridge par défaut de Docker, notamment sa plage d'adresses IP (Subnet) et les conteneurs qui y sont attachés.

```
PS C:\Users\chayma\Desktop\ing3\procedure de test> docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "b66ade411776ccd6f0dbcce7d9096f0bb84162898fcf468f17e050619a730d53",
        "Created": "2025-12-05T08:00:09.406364353Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv4": true,
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "31b106a68c203d63f4fd69a30484afb19b2fa21c334c6df4a138118459dd2fcb": {
                "Name": "web",
```

- **Crée un nouveau réseau personnalisé** de type **bridge** (par défaut) pour isoler logiquement un groupe de conteneurs.

- Liste tous les réseaux Docker disponibles sur l'hôte, confirmant la présence du réseau personnalisé monreseau et des réseaux par défaut (bridge, host, none)

```
PS C:\Users\chayma\Desktop\ing3\procedure de test> docker network create monreseau
e08fb6ee7f1f4d58d76b049ae44daa1f20dc361624fb5d624ba00eb5e3c04677
PS C:\Users\chayma\Desktop\ing3\procedure de test> docker network ls
NETWORK ID     NAME                          DRIVER    SCOPE
b66ade411776   bridge                        bridge    local
6bfcffba4a88   host                          host      local
6380ec6f44f6   mongoreplicasets_mongo-net    bridge    local
e08fb6ee7f1f   monreseau                     bridge    local
c18df402d58f   none                          null      local
```

- Lance le conteneur site et l'attache **spécifiquement** au réseau **monreseau**, lui permettant de communiquer avec d'autres conteneurs sur ce même réseau par leur nom.

```
PS C:\Users\chayma\Desktop\ing3\procedure de test> docker run -d --name site --network monreseau nginx
dfcceebd7265a040f5c0bb9bbf3dfc8ed8da1e801fbb53bd10293fe015349790
PS C:\Users\chayma\Desktop\ing3\procedure de test>
```

- Lance un deuxième conteneur site2 en le connectant au réseau **bridge par défaut** (car l'option --network est omise) et expose le port 80 du conteneur au port 8080 de l'hôte.

```
PS C:\Users\chayma\Desktop\ing3\procedure de test> docker run --name site2 -d -p 8080:80 nginx
ec32fa5943ce9003d135e5210b38cda2a18897e15e1fe4cc885a8a8c6522d49a
PS C:\Users\chayma\Desktop\ing3\procedure de test>
p1  main*  ⊗ 0 ⚠ 0
```

Docker compose

Rôle et Fonctionnement

Docker Compose est un outil qui permet de définir et d'exécuter des applications multi-conteneurs à l'aide d'un seul fichier de configuration.

- Rôle : Son rôle principal est de simplifier la gestion des environnements complexes où plusieurs services (comme une application web, une base de données et un cache) doivent démarrer ensemble et communiquer.
- Ce qu'il fait : Au lieu de lancer chaque conteneur individuellement avec de longues commandes docker run, Compose lit le fichier de configuration et automatise la création des conteneurs, la configuration de leurs réseaux, et l'attribution des volumes, le tout avec une seule commande : docker compose up.

Contenu du Fichier de Spécification (docker-compose.yml)

Le fichier de configuration, généralement nommé docker-compose.yml, est écrit au format YAML et vous permet de spécifier tout ce qui est nécessaire pour votre application.

1. Services (L'Élément Clé)

C'est la partie la plus importante. Chaque service représente un conteneur qui sera lancé. Pour chaque service, vous spécifiez :

- image : L'image Docker à utiliser (par exemple, nginx:latest ou postgres:14).
- build : L'emplacement du Dockerfile si vous devez construire l'image localement.
- ports : Le mapping des ports (par exemple, '8080:80').
- environment : Les variables d'environnement (par exemple, les mots de passe de base de données).

2. Networks (Réseaux)

Vous définissez les réseaux virtuels personnalisés.

- Rôle : Permet aux services de se connecter entre eux de manière sécurisée et d'utiliser le nom du service comme nom d'hôte (par exemple, l'application web peut se connecter à la base de données via l'hôte database).

3. Volumes

Vous spécifiez les volumes persistants ou les montages de répertoires locaux.

- Rôle : Garantit que les données importantes (comme celles des bases de données) ne sont pas perdues lorsque les conteneurs sont arrêtés, supprimés et redémarrés.

- Cette section montre le code Python de l'application, définissant une simple **API REST** utilisant le framework **Flask** et l'extension **Flask-RESTful**. La classe Product expose une route (/) qui retourne une liste statique de produits en format JSO

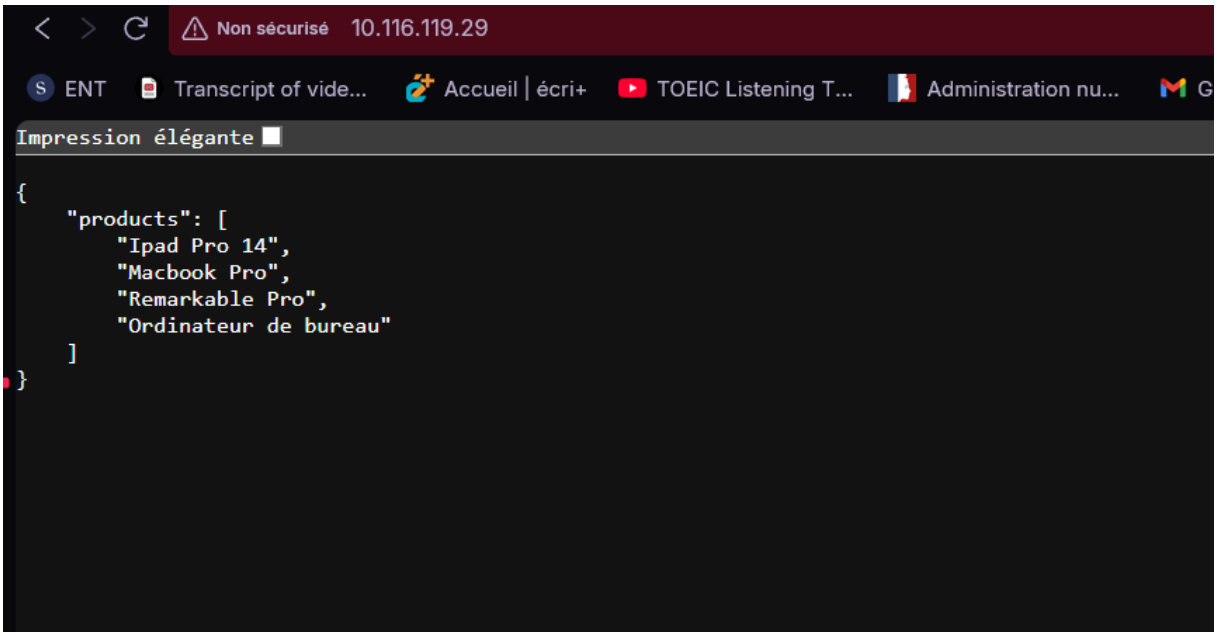
exo docker compose > product > app.py

```python
from flask import Flask
from flask_restful import Resource, Api

app = Flask(__name__)
api = Api(app)

class Product(Resource):
    def get(self):
        return {
            'products': ['Ipad Pro 14', 'Macbook Pro', 'Remarkable Pro', 'Ordinateur de bureau']
        }

api.add_resource(Product, '/')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80, debug=True)
```

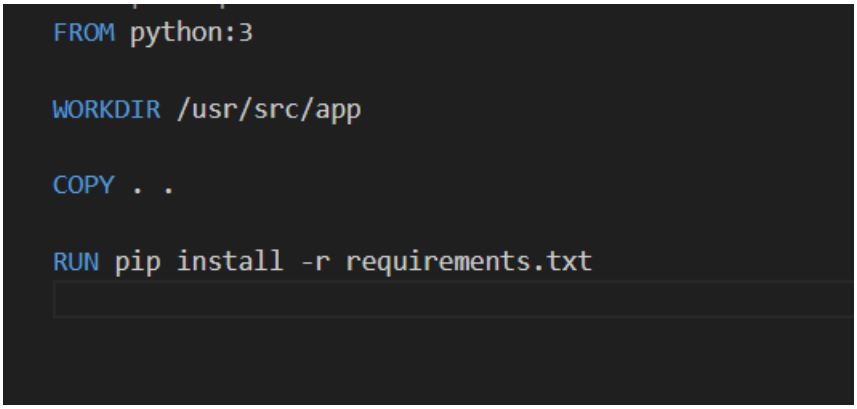
- L'exécution du script via **python app.py** démarre le serveur web Flask en mode développement (debug=true). Le serveur confirme qu'il est accessible sur l'adresse 0.0.0.0 et qu'il écoute les requêtes sur le port 80.

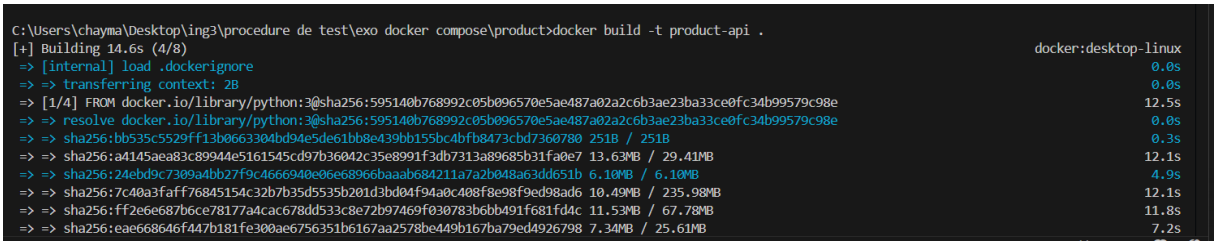
```
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\chayma\Desktop\ing3\procedure de test\exo docker compose\product> python app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://10.116.119.29:80
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 160-580-191
10.116.119.29 - - [05/Dec/2025 09:54:43] "GET / HTTP/1.1" 200 -
10.116.119.29 - - [05/Dec/2025 09:54:44] "GET /favicon.ico HTTP/1.1" 404 -
```

- La requête HTTP envoyée à l'adresse locale (10.116.119.29) réussit, et le navigateur affiche la réponse de l'API. Le résultat est la liste des produits ("products": [...]) au format **JSON**, confirmant que l'API est fonctionnelle.



- Création de l'image a partir de dockerfile





- **Définit l'API REST de backend** en Python (Flask), qui est responsable de fournir la liste des produits au format JSON.

```php
<html>
<head>
    <title>My shop</title>
</head>
<body>
    <h1>La liste des produits disponibles est :</h1>
    <ul>
        <?php
        // Récupère le JSON depuis l'API
        $json = file_get_contents('http://product-service/');
        $obj = json_decode($json);
        $products = $obj->products;

        // Boucle pour afficher chaque produit
        foreach ($products as $product) {
            echo "<li>$product</li>";
        }
        ?>
    </ul>
</body>
</html>
```

- **Démarre le serveur web Flask** en mode développement, le rendant disponible pour écouter les requêtes sur le port 80.

```yaml
services:
  product-service:
    build: ./product
    volumes:
      - ./product:/usr/src/app
    ports:
      - "5001:80"

  website:
    image: php:apache
    volumes:
      - ./website:/var/www/html
    ports:
      - "5002:80"
    depends_on:
      - product-service
```

- **Confirme l'accessibilité** de l'API Flask en affichant la réponse JSON brute de la liste des produits


localhost:5002
ENT
Transcript of vide...
Accueil | écri+
Administration nu...
Gmail
Cours : Ecri+ - Etu...
HYPER
La liste des produits disponibles est :
Ipad Pro 14
Macbook Pro
Remarkable Pro
Ordinateur de bureau