

## Créer des fichiers dockerfile

### Exemple 1 : Application Node.js simple

```
mon-app-node/
└── server.js
└── Dockerfile
```

#### server.js (serveur HTTP natif Node.js)

```
javascript
const http = require('http');

const hostname = '0.0.0.0';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/html');
  res.end(`<h1>Hello depuis Docker! </h1><p>Ceci est un serveur Node.js sans Express</p>`);
});

server.listen(port, hostname, () => {
  console.log(`Serveur démarré sur http://${hostname}:${port}`);
  console.log(`Appuyez sur Ctrl+C pour arrêter`);
});
```

#### Dockerfile simple

```
# Image de base Node.js officielle (version alpine pour la légèreté)
FROM node:18-alpine

# Définir le répertoire de travail dans le conteneur
WORKDIR /app
```

```
# Copier le fichier server.js dans le conteneur  
COPY server.js .  
  
# Exposer le port sur lequel l'application écoute  
EXPOSE 3000  
  
# Commande pour démarrer l'application  
CMD ["node", "server.js"]
```

## Commandes pour construire et exécuter

### Étape 1 : Construire l'image Docker

```
# Se placer dans le dossier contenant les fichiers  
cd mon-app-node  
  
# Construire l'image avec un tag  
docker build -t mon-serveur-node .
```

### Étape 2 : Lancer le conteneur

```
# Lancer le conteneur en mode détaché (-d) et mapper le port  
docker run -d -p 3000:3000 --name mon-serveur mon-serveur-node
```

### Étape 3 : Arrêter et nettoyer

```
# Arrêter le conteneur  
docker stop mon-serveur  
  
# Supprimer le conteneur  
docker rm mon-serveur  
  
# Supprimer l'image  
docker rmi mon-serveur-node
```

## Exemple 2 : Application Node.js simple

### Structure du projet

```
mon-app/
└── package.json
└── index.js
└── Dockerfile
```

### package.json

```
{
  "name": "mon-app",
  "version": "1.0.0",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

### index.js

```
javascript
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
  res.send('Hello Docker!');
});

app.listen(port, () => {
  console.log(`App listening at http://localhost:${port}`);
});
```

## Dockerfile

```
dockerfile
# Image de base
FROM node:18-alpine

# Répertoire de travail
WORKDIR /app

# Copie des fichiers de dépendances
COPY package*.json ./

# Installation des dépendances
RUN npm install

# Copie du code source
COPY . .

# Port exposé
EXPOSE 3000

# Commande de démarrage
CMD ["npm", "start"]
```

## Construction et exécution

```
bash
docker build -t mon-app-node .
docker run -p 3000:3000 mon-app-node
```

# Exemple 3 : Application Python Flask

## Structure

```
text
flask-app/
|__ app.py
|__ requirements.txt
└__ Dockerfile
```

## app.py

```
python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello from Flask in Docker!'

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

## requirements.txt

```
text
flask==2.3.3
```

## Dockerfile

```
dockerfile
# Image Python officielle
FROM python:3.11-slim

# Répertoire de travail
WORKDIR /app

# Copie des dépendances
```

```

COPY requirements.txt .

# Installation des dépendances
RUN pip install --no-cache-dir -r requirements.txt

# Copie du code
COPY . .

# Port exposé
EXPOSE 5000

# Commande
CMD ["python", "app.py"]

```

## Construction et exécution

```

bash
docker build -t flask-app .
docker run -p 5000:5000 flask-app

```

### Exemple 4 :

```

mon-projet-web/
├── docker-compose.yml
└── apache/
    └── public_html/
        └── index.php
└── mysql/
    └── init/
        └── init.sql (optionnel)

```

### docker-compose.yml

```

version: '3.8'

services:
  # Service MySQL
  mysql:
    image: mysql:8.0

```

```
container_name: mysql_db
restart: always
environment:
  MYSQL_ROOT_PASSWORD: rootpassword
  MYSQL_DATABASE: ma_base
  MYSQL_USER: mon_user
  MYSQL_PASSWORD: mon_password
ports:
  - "3306:3306"
volumes:
  - mysql_data:/var/lib/mysql
  - ./mysql/init:/docker-entrypoint-initdb.d
networks:
  - web-network

# Service phpMyAdmin
phpmyadmin:
  image: phpmyadmin/phpmyadmin
  container_name: phpmyadmin
  restart: always
  environment:
    PMA_HOST: mysql
    PMA_PORT: 3306
    PMA_USER: root
    PMA_PASSWORD: rootpassword
    UPLOAD_LIMIT: 100M
  ports:
    - "8080:80"
depends_on:
  - mysql
networks:
  - web-network

# Service Apache + PHP
apache:
  image: php:8.2-apache
  container_name: apache_server
  restart: always
  ports:
    - "80:80"
```

```

volumes:
- ./apache/public_html:/var/www/html
depends_on:
- mysql
networks:
- web-network

# Volumes persistants
volumes:
mysql_data:

# Réseau personnalisé
networks:
web-network:
driver: bridge

```

## Fichier de test PHP (index.php)

```

<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test MySQL + PHP + Docker</title>
    <style>
        body { font-family: Arial; margin: 40px; line-height: 1.6; }
        .success { color: green; background: #e8f5e8; padding: 10px; border-radius: 5px; }
        .error { color: red; background: #ffe8e8; padding: 10px; border-radius: 5px; }
        h1 { color: #2c3e50; }
        .info { background: #f0f0f0; padding: 10px; border-radius: 5px; }
    </style>
</head>
<body>
    <h1>Test de connexion MySQL depuis Apache/PHP</h1>

    <?php
    $host = 'mysql'; // Nom du service dans docker-compose
    $user = 'mon_user';
    $password = 'mon_password';
    $database = 'ma_base';

    echo "<div class='info'>";

```

```

echo "<h3> ↗ Informations de connexion :</h3>";
echo "Hôte: $host<br>";
echo "Base de données: $database<br>";
echo "Utilisateur: $user<br>";
echo "</div>";

try {
    // Connexion à MySQL
    $mysqli = new mysqli($host, $user, $password, $database);

    if ($mysqli->connect_error) {
        throw new Exception("Erreur de connexion: " . $mysqli->connect_error);
    }

    echo "<div class='success'> Connexion à MySQL réussie !</div>";

    // Création d'une table de test si elle n'existe pas
    $mysqli->query("CREATE TABLE IF NOT EXISTS test (
        id INT AUTO_INCREMENT PRIMARY KEY,
        message VARCHAR(255),
        date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )");

    // Insertion d'un message de test
    $mysqli->query("INSERT INTO test (message) VALUES ('Test depuis Apache/PHP')");

    // Récupération des derniers messages
    $result = $mysqli->query("SELECT * FROM test ORDER BY date DESC LIMIT 5");

    if ($result->num_rows > 0) {
        echo "<h3> ↗ Derniers messages dans la base :</h3>";
        echo "<ul>";
        while ($row = $result->fetch_assoc()) {
            echo "<li>" . $row['date'] . " " . $row['message'] . "</li>";
        }
        echo "</ul>";
    }

    $mysqli->close();

} catch (Exception $e) {
    echo "<div class='error'> Erreur : " . $e->getMessage() . "</div>";
}
?>

<hr>
<p>

```

 Accéder à phpMyAdmin : <a href="http://localhost:8080" target="\_blank">http://localhost:8080</a><br> Identifiants phpMyAdmin : root / rootpassword</p></body></html>

## Fichier SQL d'initialisation (optionnel)

```
-- Création d'une table utilisateurs
CREATE TABLE IF NOT EXISTS utilisateurs (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    date_inscription TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Insertion de données de test
INSERT INTO utilisateurs (nom, email) VALUES
('Jean Dupont', 'jean@example.com'),
('Marie Martin', 'marie@example.com'),
('Pierre Durand', 'pierre@example.com');

-- Création d'une table produits
CREATE TABLE IF NOT EXISTS produits (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nom VARCHAR(200) NOT NULL,
    prix DECIMAL(10,2) NOT NULL,
    stock INT DEFAULT 0
);

-- Insertion de produits de test
INSERT INTO produits (nom, prix, stock) VALUES
('Ordinateur portable', 899.99, 10),
('Souris sans fil', 29.99, 50),
('Clavier mécanique', 79.99, 25);
```

# **Commandes pour gérer le déploiement**

## **Démarrer tous les services**

```
# Démarrer en mode détaché (arrière-plan)  
docker-compose up -d
```

```
# Ou démarrer avec Les Logs en direct  
docker-compose up
```