



01/02/2025

# Mise en œuvre d'une API Gateway avec NGINX

Mettre en place une API Gateway à l'aide de NGINX pour centraliser et sécuriser l'accès à plusieurs microservices d'une application web



MAHDI KELLOUCH  
ISMO TETOUAN

# Table des matières

1. Étude de Cas : Système de Gestion de Commandes en Ligne .....	2
2. Plan de l'Atelier.....	2
a. Préparation de l'Environnement.....	2
b. Déploiement des Microservices (Simulés avec Express.js).....	3
• Service d'authentification (auth-service).....	3
• Service des commandes (order-service) .....	3
• Service des menus (menu-service) .....	4
c. Configuration de l'API Gateway avec NGINX .....	4
d. Déploiement avec Docker Compose.....	5
e. Test de l'API Gateway .....	5
1- Lancer les services : .....	5
2- Tester l'API Gateway avec Postman ou Curl : .....	5
3. Exercices à réaliser .....	6

# Activité : Mise en œuvre d'une API Gateway avec NGINX

## 1. Étude de Cas : Système de Gestion de Commandes en Ligne

Une entreprise de livraison de repas souhaite centraliser l'accès à ses différents services via une **API Gateway**. L'architecture de l'application repose sur plusieurs microservices :

1. **Service d'authentification** (auth-service) : Gestion des utilisateurs et des tokens d'authentification.
2. **Service des commandes** (order-service) : Permet de passer et suivre des commandes.
3. **Service des menus** (menu-service) : Permet d'afficher les plats disponibles.

Actuellement, chaque service expose ses propres API sur différents ports, ce qui oblige les clients à connaître leurs adresses spécifiques.

Pour simplifier et sécuriser l'accès, l'entreprise décide d'implémenter une **API Gateway avec NGINX**, qui servira d'unique point d'entrée pour toutes les requêtes.

## 2. Plan de l'Atelier

### a. Préparation de l'Environnement

1. Assurez-vous que Docker est installé.
2. Créer un répertoire pour le projet avec la structure suivante :

```
▼ CN-ACTIVITE-API-GATEWAY
  ▼ auth-service
    🐳 Dockerfile
    {} package.json
    JS server.js
  ▼ menu-service
    🐳 Dockerfile
    {} package.json
    JS server.js
  ▼ nginx
    ⚙️ nginx.conf
  ▼ order-service
    🐳 Dockerfile
    {} package.json
    JS server.js
  🚢 docker-compose.yml
```

## b. Déploiement des Microservices (Simulés avec Express.js)

Nous allons simuler les microservices en créant trois petits serveurs avec Node.js et Express.

- Service d'authentification (auth-service)

- Créer le fichier auth-service/server.js :

```
const express = require('express');
const app = express();
app.use(express.json());

app.post('/login', (req, res) => {
  res.json({ token: "fake-jwt-token" });
});

app.listen(5001, () => console.log("Auth Service running on port 5001"));
```

- Créer le fichier auth-service/ Dockerfile:

```
FROM node:14
WORKDIR /app
COPY package.json .
COPY server.js .
RUN npm install
EXPOSE 5001
CMD ["node", "server.js"]
```

- Service des commandes (order-service)

- Créer le fichier order-service/server.js :

```
const express = require('express');

const app = express();
app.use(express.json());

app.get('/orders', (req, res) => {
  res.json([{ id: 1, meal: "Pizza", status: "delivered" }]);
});

app.listen(5002, () => console.log("Order Service running on port 5002"));
```

- Créer le fichier order-service/ Dockerfile:

```
FROM node:14
WORKDIR /app
COPY package.json .
COPY server.js .
RUN npm install
EXPOSE 5002
CMD ["node", "server.js"]
```

- Service des menus (menu-service)

- Créer le fichier menu-service/server.js :

```
const express = require('express');

const app = express();
app.use(express.json());

app.get('/menu', (req, res) => {
  res.json([ { id: 1, name: "Pizza", price: 10 } ]);
});

app.listen(5003, () => console.log("Menu Service running on port 5003"));
```

- Créer le fichier menu-service/ Dockerfile:

```
FROM node:14
WORKDIR /app
COPY package.json .
COPY server.js .
RUN npm install
EXPOSE 5003
CMD ["node", "server.js"]
```

### c. Configuration de l'API Gateway avec NGINX

Créer un fichier nginx/nginx.conf :

```
events { }
```

```
http {
    upstream auth_service {
        server auth-service:5001;
    }

    upstream order_service {
        server order-service:5002;
    }

    upstream menu_service {
        server menu-service:5003;
    }

    server {
        listen 80;

        location /api/auth/ {
            proxy_pass http://auth_service;
        }

        location /api/orders/ {
            proxy_pass http://order_service;
            proxy_set_header Authorization $http_authorization;
        }
    }
}
```

```

    }

    location /api/menu/ {
        proxy_pass http://menu_service/;
    }
}
}

```

#### d. Déploiement avec Docker Compose

Créer un fichier `./docker-compose.yml` pour gérer tous les services.

```

version: '3'
services:
  nginx:
    image: nginx:latest
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
    ports:
      - "8080:80"
    depends_on:
      - auth-service
      - order-service
      - menu-service

  auth-service:
    build: ./auth-service
    ports:
      - "5001:5001"

  order-service:
    build: ./order-service
    ports:
      - "5002:5002"

  menu-service:
    build: ./menu-service
    ports:
      - "5003:5003"

```

#### e. Test de l'API Gateway

1- Lancer les services :

```
docker-compose up -build
```

2- Tester l'API Gateway avec Postman ou Curl :

- **Authentification :**

Invoke-WebRequest -Method POST -Uri <http://localhost:8080/api/auth/login>

(Invoke-WebRequest -Method POST -Uri <http://localhost:8080/api/auth/login>).content

- **Lister les commandes :**

Invoke-WebRequest -Method GET -Uri "<http://localhost:8080/api/orders/orders> " -Headers @{"Authorization="fake-jwt-token"}

- **Voir le menu :**

Invoke-WebRequest -Method GET -Uri <http://localhost:8080/api/menu/menu>

### 3. Exercices à réaliser

1. **Ajouter un endpoint `/api/orders/ :id`** permettant d'obtenir une commande par ID.
2. **Ajouter une authentification basique** sur les commandes, en vérifiant qu'un token est présent dans le header avant de renvoyer une réponse.