

# EVIMERIA (anciennement JelleShop)

---

Boutique e-commerce moderne créée avec Django, React et Cloudinary.

## Prérequis

- Docker et Docker Compose pour le développement local
- Un compte Railway pour le déploiement
- Un compte Cloudinary pour la gestion des médias

## Structure du Projet

```
evimeria/
├── backend/                # API Django
│   ├── Dockerfile         # Dockerfile du backend
│   ├── jaelleshop/        # Configurations du projet
│   ├── products/          # App pour les produits
│   ├── users/             # App pour les utilisateurs
│   └── ...
├── frontend/              # Application React
│   ├── Dockerfile         # Dockerfile du frontend
│   ├── nginx.conf         # Configuration Nginx
│   └── ...
├── docker-compose.yml     # Configuration Docker Compose
├── railway.toml           # Configuration Railway
└── ...
```

## Déploiement sur Railway

### 1. Préparation du code

Assurez-vous que votre dépôt contient tous les fichiers suivants:

- `backend/Dockerfile`
- `frontend/Dockerfile`
- `frontend/nginx.conf`
- `docker-compose.yml`
- `railway.toml`
- `.dockerignore`

### 2. Configuration des variables d'environnement

Dans Railway, configurez les variables d'environnement suivantes:

```

# Base de données PostgreSQL
DATABASE_URL=postgresql://${PGUSER}:${POSTGRES_PASSWORD}@${PGHOST}:${PGPORT}/${PGDATABASE}
POSTGRES_PASSWORD=votre_mot_de_passe
POSTGRES_USER=postgres
POSTGRES_DB=railway

# Configuration Django
DEBUG=False
SECRET_KEY=votre_secret_key
ALLOWED_HOSTS=*.up.railway.app,localhost,127.0.0.1

# Cloudinary
CLOUDINARY_CLOUD_NAME=dmcaguchx
CLOUDINARY_API_KEY=votre_api_key
CLOUDINARY_API_SECRET=votre_api_secret

# Configuration de build
NODE_OPTIONS=--max_old_space_size=465
PIP_NO_CACHE_DIR=true
PYTHONUNBUFFERED=1
NODE_ENV=production
NPM_CONFIG_PRODUCTION=false

# Port
PORT=8000

```

### 3. Déploiement sur Railway

1. Connectez-vous à Railway et créez un nouveau projet
2. Choisissez "Deploy from GitHub repo"
3. Sélectionnez votre dépôt GitHub
4. Railway détectera automatiquement les fichiers Docker et déploiera votre application

### 4. Configuration des volumes Railway

Railway créera automatiquement les volumes suivants, configurés dans le fichier `railway.toml`:

- `backend_static`: Pour les fichiers statiques Django
- `backend_media`: Pour les médias uploadés
- `postgres_data`: Pour les données PostgreSQL persistantes

## Développement local avec Docker

```

# Démarrer l'application en mode développement
docker-compose up

# Reconstruire l'application après des modifications
docker-compose up --build

```

```
# Exécuter des commandes dans le conteneur backend
docker-compose exec backend python manage.py createsuperuser
```

## Maintenance

### Migration de la base de données

```
docker-compose exec backend python manage.py makemigrations
docker-compose exec backend python manage.py migrate
```

### Création d'un utilisateur admin

```
docker-compose exec backend python manage.py createsuperuser
```

### Sauvegarde et restauration de la base de données

```
# Sauvegarde
docker-compose exec db pg_dump -U postgres railway > backup.sql

# Restauration
cat backup.sql | docker-compose exec -T db psql -U postgres railway
```