HR

Guide de Deploiement

HR_ONIAN — Systeme de Gestion des Ressources Humaines

Docker | Django 5.0 | PostgreSQL 15 | Nginx | Let's Encrypt

Version 1.0 • Fevrier 2026 • ONIAN-EasyM

Guide de Deploiement HR_ONIAN

Table des matieres

---

1. Prerequis

Logiciels requis

Logiciel	Version minimale	Verification
Docker	>= 20.10	`docker --version`
Docker Compose	>= 2.0	`docker compose version`
Git	>= 2.0	`git --version`

Ressources serveur recommandees

Ressource	Minimum	Recommande
RAM	2 Go	4 Go
CPU	1 coeur	2 coeurs
Disque	10 Go	20 Go

Ports requis

Port	Service	Description
80	Nginx	HTTP (redirige vers HTTPS en production)
443	Nginx	HTTPS (production)
8000	Gunicorn	Application Django (interne, non expose en prod)
5432	PostgreSQL	Base de donnees (interne, non expose en prod)

---

2. Demarrage rapide

Trois commandes suffisent pour demarrer l'application :

```
# 1. Cloner le projet
git clone <url-du-depot> HR_ONIAN
cd HR_ONIAN

# 2. Configurer l'environnement
cp .env.docker .env.docker.local
# Editer .env.docker.local si necessaire (mot de passe BDD, SECRET_KEY, etc.)

# 3. Lancer l'application
docker compose up -d --build
```

L'application est accessible a l'adresse : **http://localhost:8000**

Premier lancement avec donnees par defaut

Pour charger les donnees de reference et l'employe administrateur (MT000001) :

```
LOAD_DEFAULT_DATA=true LOAD_DEFAULT_EMPLOYEE=true docker compose up -d --build
```

---

3. Configuration

Variables d'environnement

Toute la configuration se fait via le fichier `.env.docker`. Voici la liste complete des variables :

Django

Variable	Description	Defaut	Obligatoire
`SECRET_KEY`	Cle secrete Django (unique par environnement)	-	Oui
`DEBUG`	Mode debug (`True` / `False`)	`False`	Oui
`ALLOWED_HOSTS`	Domaines autorises (separes par des virgules)	-	Oui

Base de donnees

Variable	Description	Defaut
`DB_NAME`	Nom de la base de donnees	`hrapp`
`DB_USER`	Utilisateur PostgreSQL	`hr`
`DB_PASSWORD`	Mot de passe PostgreSQL	-
`DB_HOST`	Hote de la base de donnees	`db` (nom du service Docker)
`DB_PORT`	Port PostgreSQL	`5432`

PostgreSQL (conteneur)

Variable	Description	Defaut
`POSTGRES_DB`	Nom de la base a creer	`hrapp`
`POSTGRES_USER`	Utilisateur a creer	`hr`
`POSTGRES_PASSWORD`	Mot de passe	-

> **Important** : `DB_PASSWORD` et `POSTGRES_PASSWORD` doivent avoir la **meme valeur**.

Email (production)

Variable	Description	Exemple
`EMAIL_HOST`	Serveur SMTP	`smtp.gmail.com`
`EMAIL_PORT`	Port SMTP	`587`
`EMAIL_USE_TLS`	Utiliser TLS	`True`
`EMAIL_HOST_USER`	Adresse email	`hr@votre-domaine.com`
`EMAIL_HOST_PASSWORD`	Mot de passe applicatif	-
`DEFAULT_FROM_EMAIL`	Expediteur par defaut	`ONIAN-EasyM <noreply@domaine.com>`

Donnees par defaut

Variable	Description	Defaut
`LOAD_DEFAULT_DATA`	Charger les donnees de reference au demarrage	`false`
`LOAD_DEFAULT_EMPLOYEE`	Charger l'employe MT000001 au demarrage	`false`

Superutilisateur (optionnel)

Variable	Description
`DJANGO_SUPERUSER_USERNAME`	Nom d'utilisateur admin
`DJANGO_SUPERUSER_EMAIL`	Email admin
`DJANGO_SUPERUSER_PASSWORD`	Mot de passe admin

Gunicorn

Variable	Description	Defaut
`GUNICORN_BIND`	Adresse d'ecoute	`0.0.0.0:8000`
`GUNICORN_WORKERS`	Nombre de workers	`(2 x CPU) + 1`

4. Deploiement en developpement

Lancer l'environnement de developpement

```
# Demarrer tous les services
docker compose up --build

# Ou en arriere-plan
docker compose up -d --build
```

En mode developpement :
- Le serveur de developpement Django (`runserver`) est utilise a la place de Gunicorn
- Le code source est monte en volume pour le **rechargement automatique** (live reload)
- `DEBUG=True` est active
- La base de donnees PostgreSQL est accessible sur le port **5433** de l'hote

Commandes utiles en developpement

```
# Executer une commande Django dans le conteneur
docker compose exec web python manage.py <commande>

# Exemples :
docker compose exec web python manage.py createsuperuser
docker compose exec web python manage.py makemigrations
docker compose exec web python manage.py migrate
docker compose exec web python manage.py shell

# Acceder au shell du conteneur
docker compose exec web bash

# Voir les logs en temps reel
docker compose logs -f web
```

Arreter l'environnement

```
# Arreter les conteneurs (les donnees sont conservees)
docker compose down

# Arreter et supprimer toutes les donnees (volumes)
docker compose down -v
```

---

5. Deploiement en production

Etape 1 : Preparer le serveur

```
# Se connecter au serveur
ssh utilisateur@votre-serveur

# Installer Docker (si pas deja installe)
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER

# Installer Docker Compose plugin
sudo apt-get install docker-compose-plugin
```

Etape 2 : Cloner le projet

```
cd /opt
sudo git clone <url-du-depot> HR_ONIAN
cd HR_ONIAN
sudo chown -R $USER:$USER .
```

Etape 3 : Configurer l'environnement de production

```
# Copier le fichier d'environnement
cp .env.docker .env.docker

# Editer avec les valeurs de production
nano .env.docker
```

Modifications obligatoires pour la production :

```
# Generer une nouvelle SECRET_KEY :
# python3 -c "from django.core.management.utils import get_random_secret_key; print(get_random_secret_key())"
SECRET_KEY=votre-cle-secrete-generee-ici

DEBUG=False
ALLOWED_HOSTS=votre-domaine.com,www.votre-domaine.com

DB_PASSWORD=un-mot-de-passe-fort-et-unique
POSTGRES_PASSWORD=un-mot-de-passe-fort-et-unique

SITE_URL=https://votre-domaine.com

# Email SMTP
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=votre-email@gmail.com
EMAIL_HOST_PASSWORD=votre-mot-de-passe-applicatif
```

Etape 4 : Lancer en production

```
# Premier lancement (avec donnees par defaut)
LOAD_DEFAULT_DATA=true LOAD_DEFAULT_EMPLOYEE=true \
  docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build

# Lancements suivants
docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build
```

Etape 5 : Verifier le deploiement

```
# Verifier que les conteneurs tournent
docker compose ps

# Verifier le health check
curl http://localhost/health/

# Verifier les logs
docker compose logs -f web
```

---

6. Certificat SSL (Let's Encrypt)

Etape 1 : Obtenir le certificat initial

```
# S'assurer que le domaine pointe vers le serveur (DNS A record)
# S'assurer que le port 80 est ouvert

# Obtenir le certificat
docker compose -f docker-compose.yml -f docker-compose.prod.yml \
  run --rm certbot certonly --webroot \
  --webroot-path=/var/lib/letsencrypt \
  -d votre-domaine.com -d www.votre-domaine.com \
  --email votre-email@gmail.com --agree-tos --no-eff-email
```

Etape 2 : Activer HTTPS dans Nginx

Editer le fichier `deploy/nginx/default.conf` :

1. Decommenter le bloc `server` HTTPS (port 443)
2. Remplacer `votre-domaine.com` par votre domaine reel
3. Modifier le bloc HTTP (port 80) pour rediriger vers HTTPS

```
# Redemarrer Nginx
docker compose -f docker-compose.yml -f docker-compose.prod.yml restart nginx
```

Etape 3 : Renouvellement automatique

Le conteneur `certbot` renouvelle automatiquement les certificats toutes les 12 heures.
Aucune action manuelle n'est necessaire.

Pour verifier le renouvellement :

```
docker compose -f docker-compose.yml -f docker-compose.prod.yml \
  run --rm certbot renew --dry-run
```

---

7. Gestion de la base de donnees

Appliquer les migrations

```
docker compose exec web python manage.py migrate
```

Creer les migrations apres modification des modeles

```
docker compose exec web python manage.py makemigrations
docker compose exec web python manage.py migrate
```

Acceder au shell PostgreSQL

```
docker compose exec db psql -U hr -d hrapp
```

Acceder au shell Django

```
docker compose exec web python manage.py shell
```

---

8. Donnees par defaut

Le projet inclut des commandes de gestion pour charger des donnees de reference.

Donnees de reference (departements, postes, types d'absence, etc.)

```
# Charger les donnees
docker compose exec web python manage.py charger_donnees

# Simulation (sans ecriture)
docker compose exec web python manage.py charger_donnees --dry-run

# Forcer la mise a jour des donnees existantes
docker compose exec web python manage.py charger_donnees --force
```

Employe administrateur par defaut (MT000001)

```
# Charger l'employe
docker compose exec web python manage.py charger_employe

# Avec un nouveau mot de passe
docker compose exec web python manage.py charger_employe --password MonMotDePasse123

# Simulation
docker compose exec web python manage.py charger_employe --dry-run
```

Extraire les donnees (pour creer de nouveaux jeux de donnees)

```
# Extraire les donnees de reference
docker compose exec web python manage.py extraire_donnees

# Extraire un employe specifique
docker compose exec web python manage.py extraire_employe MT000002
```

---

9. Sauvegardes et restauration

Sauvegarder la base de donnees

```
# Sauvegarde manuelle
docker compose exec db pg_dump -U hr hrapp | gzip > backups/hrapp_$(date +%Y%m%d_%H%M%S).sql.gz

# Ou via le script existant (depuis le conteneur web)
docker compose exec web bash scripts/backup_postgresql.sh
```

Restaurer la base de donnees

```
# Lister les sauvegardes disponibles
ls -la backups/

# Restaurer depuis une sauvegarde
gunzip -c backups/hrapp_20260221_120000.sql.gz | docker compose exec -T db psql -U hr -d hrapp
```

Sauvegarder les fichiers media

```
# Sauvegarde manuelle
docker compose exec web tar -czf /app/backups/media_$(date +%Y%m%d_%H%M%S).tar.gz -C /app media/
```

Sauvegardes automatiques (cron sur le serveur hote)

Ajouter au crontab du serveur hote (`crontab -e`) :

```
# Sauvegarde BDD - Tous les jours a 2h du matin
0 2 * * * cd /opt/HR_ONIAN && docker compose exec -T db pg_dump -U hr hrapp | gzip > backups/hrapp_$(date +\%Y\%m\%d_\%H\%M\
%S).sql.gz

# Nettoyage des sauvegardes > 7 jours
0 3 * * 0 find /opt/HR_ONIAN/backups/ -name "*.gz" -mtime +7 -delete
```

---

10. Monitoring et logs

Verifier l'etat des conteneurs

```
# Etat de tous les services
docker compose ps

# Utilisation des ressources
docker stats
```

Endpoint de sante

L'application expose un endpoint de verification :

```
curl http://localhost/health/
# Reponse : {"status": "ok", "database": "connected"}
```

Consulter les logs

```
# Tous les services
docker compose logs -f

# Un service specifique
docker compose logs -f web
docker compose logs -f db
docker compose logs -f nginx

# Les 100 dernieres lignes
docker compose logs --tail=100 web
```

Logs applicatifs Django

Les logs Django sont stockes dans le volume `logs` :

```
# Voir le log principal
docker compose exec web cat logs/hr_onian.log

# Voir les erreurs
docker compose exec web cat logs/errors.log

# Suivre en temps reel
docker compose exec web tail -f logs/hr_onian.log
```

---

11. Mises a jour

Mettre a jour l'application

```
# 1. Recuperer les dernieres modifications
cd /opt/HR_ONIAN
git pull origin main

# 2. Reconstruire et relancer
docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build

# Le script d'entrypoint applique automatiquement :
# - Les migrations de base de donnees
# - La collecte des fichiers statiques
```

Mettre a jour un seul service

```
# Reconstruire uniquement le service web
docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build web
```

Rollback (retour en arriere)

```
# 1. Revenir au commit precedent
git checkout <commit-precedent>

# 2. Reconstruire
docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build
```

---

12. Deploiement sans Docker (methode traditionnelle)

Si vous preferez deployer sans Docker, le projet inclut les fichiers necessaires :

Fichiers disponibles

Fichier	Description
`deploy/deploy.sh`	Script de deploiement automatise (7 etapes)
`deploy/nginx.conf`	Configuration Nginx avec SSL
`deploy/hr_onian.service`	Service systemd pour Gunicorn
`gunicorn.conf.py`	Configuration Gunicorn

Procedure

```
# 1. Installer les prerequis
sudo apt-get install python3.12 python3.12-venv postgresql nginx certbot

# 2. Creer l'environnement virtuel
python3.12 -m venv venv
source venv/bin/activate

# 3. Configurer l'environnement
cp .env.production .env.local
nano .env.local

# 4. Lancer le script de deploiement
bash deploy/deploy.sh

# 5. Configurer Nginx
sudo cp deploy/nginx.conf /etc/nginx/sites-available/hr_onian
sudo ln -s /etc/nginx/sites-available/hr_onian /etc/nginx/sites-enabled/
sudo nginx -t && sudo systemctl reload nginx

# 6. Configurer le service systemd
sudo cp deploy/hr_onian.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable hr_onian
sudo systemctl start hr_onian

# 7. Obtenir un certificat SSL
sudo certbot --nginx -d votre-domaine.com
```

---

13. Depannage

Le conteneur web ne demarre pas

```
# Verifier les logs
docker compose logs web

# Causes frequentes :
# - PostgreSQL pas encore pret → le script attend automatiquement (30 tentatives)
# - SECRET_KEY manquante → verifier .env.docker
# - Erreur de migration → verifier les logs pour l'erreur specifique
```

Erreur "Could not find config for 'default' in STORAGES"

Cette erreur est resolue. Si elle reapparait, verifier que `settings.py` contient :

```
STORAGES = {
    "default": {
        "BACKEND": "django.core.files.storage.FileSystemStorage",
    },
    "staticfiles": {
        "BACKEND": "whitenoise.storage.CompressedManifestStaticFilesStorage",
    },
}
```

Les fichiers statiques ne s'affichent pas

```
# Recollecte des fichiers statiques
docker compose exec web python manage.py collectstatic --noinput --clear

# Redemarrer Nginx
docker compose restart nginx
```

La base de donnees n'est pas accessible

```
# Verifier que le conteneur db tourne
docker compose ps db

# Verifier les logs PostgreSQL
docker compose logs db

# Tester la connexion
docker compose exec db psql -U hr -d hrapp -c "SELECT 1;"
```

Erreur de permission sur les volumes

```
# Verifier les permissions
docker compose exec web ls -la /app/media/
docker compose exec web ls -la /app/logs/

# Corriger si necessaire (depuis l'hote)
docker compose exec --user root web chown -R appuser:appgroup /app/media /app/logs
```

Reinitialiser completement l'environnement

```
# ATTENTION : Cela supprime TOUTES les donnees !
docker compose down -v
docker compose up -d --build
```

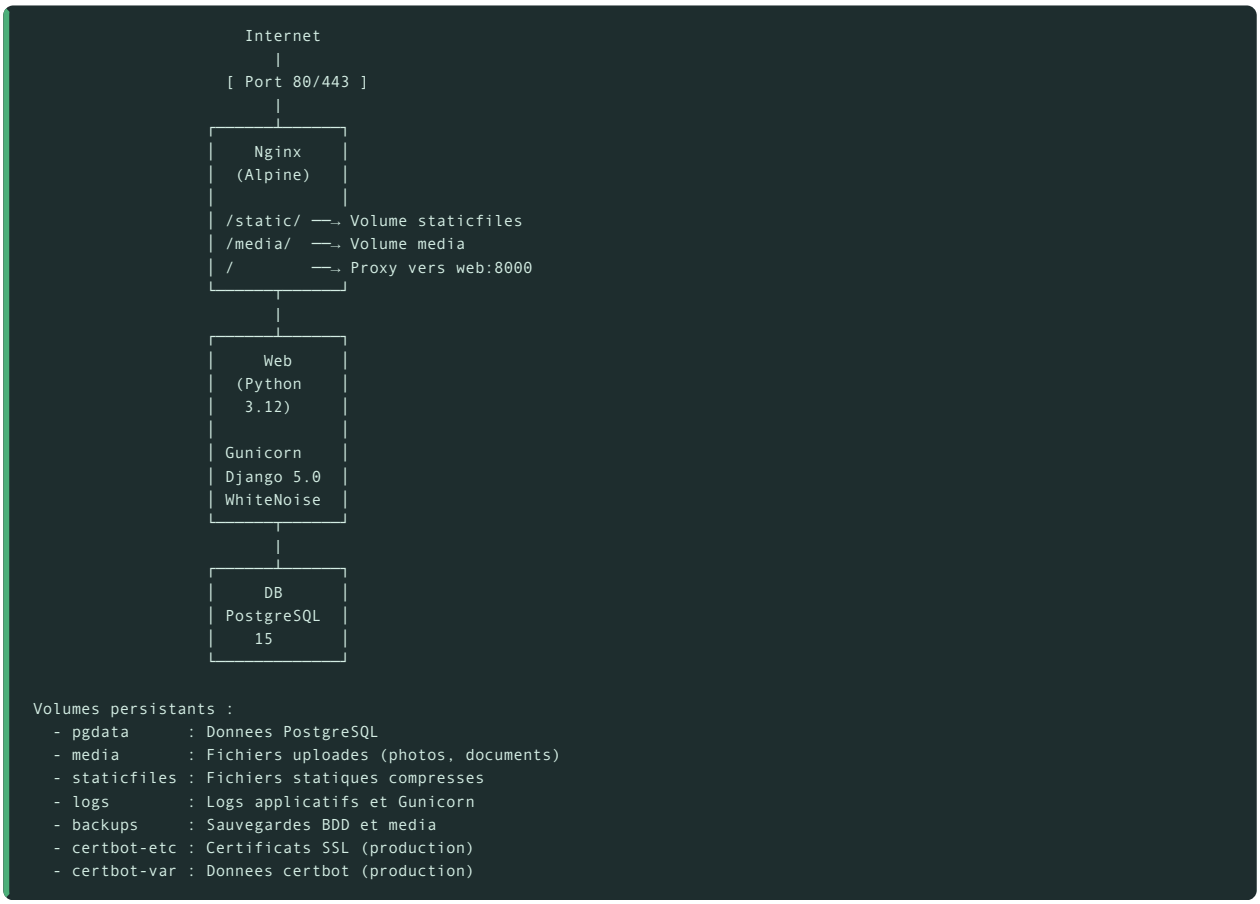
Problemes de memoire

```
# Verifier l'utilisation memoire
docker stats

# Reduire le nombre de workers Gunicorn
# Dans .env.docker :
GUNICORN_WORKERS=2
```

---

14. Architecture technique



Flux de demarrage

```
1. Docker Compose lance les conteneurs
2. PostgreSQL demarre et accepte les connexions (healthcheck)
3. Le conteneur web attend PostgreSQL (docker-entrypoint.sh)
4. Migrations appliquees automatiquement
5. Fichiers statiques collectes
6. Donnees par defaut chargees (si LOAD_DEFAULT_DATA=true)
7. Gunicorn demarre avec (2 x CPU + 1) workers
8. Nginx demarre et route le trafic vers Gunicorn
```

Ports internes (reseau Docker)

Service	Port interne	Expose sur l'hote
db	5432	5433 (dev) / non (prod)
web	8000	8000 (dev) / non (prod)
nginx	80, 443	80, 443

Commandes de reference rapide

```
# === DEVELOPPEMENT ===
docker compose up --build                   # Demarrer (foreground)
docker compose up -d --build                # Demarrer (arriere-plan)
docker compose down                         # Arreter
docker compose logs -f web                  # Voir les logs

# === PRODUCTION ===
docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d --build
docker compose -f docker-compose.yml -f docker-compose.prod.yml down
docker compose -f docker-compose.yml -f docker-compose.prod.yml logs -f

# === COMMANDES DJANGO ===
docker compose exec web python manage.py migrate
docker compose exec web python manage.py createsuperuser
docker compose exec web python manage.py shell
docker compose exec web python manage.py collectstatic --noinput

# === BASE DE DONNEES ===
docker compose exec db psql -U hr -d hrapp
docker compose exec db pg_dump -U hr hrapp > backup.sql

# === MAINTENANCE ===
docker compose ps                           # Etat des conteneurs
docker stats                                # Utilisation ressources
curl http://localhost/health/                # Health check
```