🚀 Guide de Déploiement Production - RAG System

📋 Table des Matières

Prérequis

Matériel Requis

Minimum (pour test):

- CPU: 8 cores
- RAM: 32 GB
- Stockage: 100 GB SSD
- GPU: Optionnel (améliore performances de 5-10x)

Recommandé (pour production):

- CPU: 16 cores
- RAM: 64 GB
- Stockage: 500 GB NVMe SSD
- GPU: NVIDIA RTX 4090 / A100 (24GB VRAM) pour fine-tuning

Logiciels

```bash
```

```
# Ubuntu 22.04 LTS
- Docker 24+
- Docker Compose 2.20+
- Python 3.10+
- PostgreSQL 15
- Redis 7
- Nginx 1.24
```

Installation

1. Cloner le Repository

```bash
git clone https://github.com/votrecompagnie/rag-system.git
cd rag-system
```

2. Configuration Environnement

```bash
# Copier le fichier d'exemple
cp .env.example .env

# Générer des secrets sécurisés
openssl rand -base64 32  # Pour SECRET_KEY
openssl rand -base64 32  # Pour ENCRYPTION_KEY

# Éditer .env avec vos valeurs
nano .env
```

3. Télécharger le Modèle LLM

```bash
# Créer le dossier
mkdir -p data/models

# Télécharger Mistral-7B (4-bit GGUF)
wget -O data/models/mistral-7b-instruct-v0.3.Q4_K_M.gguf \
  https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.3-GGUF/resolve/main/mistral-7b-instruct-v0.3.Q4_K_M.gguf

# Vérifier l'intégrité (environ 4.1 GB)
ls -lh data/models/
```

4. Build Docker Images

```bash
make build
# Ou manuellement:
docker-compose -f docker-compose.prod.yml build
```

Configuration

Configuration PostgreSQL

```sql
-- backend/alembic/versions/001_initial.sql
CREATE TABLE IF NOT EXISTS users (
    id VARCHAR PRIMARY KEY,
    email VARCHAR UNIQUE NOT NULL,
    hashed_password VARCHAR NOT NULL,
    role VARCHAR DEFAULT 'user',
    tenant_id VARCHAR,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE TABLE IF NOT EXISTS documents (
    id VARCHAR PRIMARY KEY,
    user_id VARCHAR REFERENCES users(id),
    filename VARCHAR NOT NULL,
    status VARCHAR DEFAULT 'pending',
    num_chunks INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_documents_user ON documents(user_id);
CREATE INDEX idx_documents_status ON documents(status);
```

Configuration Redis Cache

```conf
# Ajouter à redis.conf
maxmemory 2gb
maxmemory-policy allkeys-lru
appendonly yes
```

Configuration Prometheus

```yaml
# monitoring/prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'rag_api'
    static_configs:
      - targets: ['api:8000']
```

Déploiement

1. Déploiement Local

```bash
# Démarrer tous les services
make up

# Vérifier les logs
make logs

# Accéder aux interfaces:
# - API Docs: http://localhost:8000/docs
# - Grafana: http://localhost:3000 (admin/password)
# - RabbitMQ: http://localhost:15672 (guest/guest)
```

2. Initialisation Base de Données

```bash
# Exécuter les migrations
docker-compose exec api alembic upgrade head

# Créer un utilisateur admin
docker-compose exec api python scripts/create_admin.py
```

3. Tests de Santé

```bash
```

```
# Health check API
curl http://localhost:8000/api/v1/health

# Test ingestion
curl -X POST "http://localhost:8000/api/v1/documents/upload" \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -F "file=@test.pdf"

# Test query
curl -X POST "http://localhost:8000/api/v1/query" \
  -H "Authorization: Bearer YOUR_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"query": "What is RAG?", "top_k": 5}'
```

4. Déploiement Production (AWS)

```
bash

# Configuration Terraform
cd infrastructure/terraform
terraform init
terraform plan
terraform apply

# Déploiement Kubernetes
cd infrastructure/kubernetes
kubectl apply -f namespace.yaml
kubectl apply -f deployments/
kubectl apply -f services/
kubectl apply -f ingress/

# Vérifier le déploiement
kubectl get pods -n rag-system
kubectl logs -f deployment/rag-api -n rag-system
```

Monitoring

1. Métriques Prometheus

Métriques clés:

- `rag_queries_total`: Total des requêtes
- `rag_retrieval_latency_seconds`: Latence de retrieval
- `rag_generation_latency_seconds`: Latence de génération

- `rag_total_latency_seconds`: Latence totale
- `rag_index_size`: Taille de l'index vectoriel

2. Dashboards Grafana

Importer les dashboards pré-configurés:

```bash
# Copier les dashboards
cp monitoring/grafana/dashboards/*.json /var/lib/grafana/dashboards/

# Accéder à Grafana: http://localhost:3000
# Login: admin / votre_mot_de_passe
```

Dashboards disponibles:

1. **Overview**: Métriques système globales
2. **Performance**: Latences et throughput
3. **Quality**: Taux de succès, feedback utilisateurs
4. **Resources**: CPU, RAM, GPU utilization

3. Alertes

```yaml
# monitoring/alerts.yml
groups:
  - name: rag_alerts
    rules:
      - alert: HighLatency
        expr: histogram_quantile(0.95, rag_total_latency_seconds) > 10
        for: 5m
        annotations:
          summary: "95th percentile latency > 10s"

      - alert: HighErrorRate
        expr: rate(rag_queries_total{status="error"}[5m]) > 0.05
        for: 5m
        annotations:
          summary: "Error rate > 5%"
```

Maintenance

1. Backups Automatiques

```bash
# Configurer cron pour backups quotidiens
crontab -e

# Ajouter:
0 2 * * * /app/scripts/backup.sh >> /var/log/backup.log 2>&1
```

2. Rotation des Logs

```bash
# /etc/logrotate.d/rag-system
/app/logs/*.log {
    daily
    rotate 30
    compress
    delaycompress
    notifempty
    create 0640 www-data www-data
    sharedscripts
    postrotate
        docker-compose restart api
    endscript
}
```

3. Mise à Jour du Modèle

```bash
# Télécharger nouveau modèle
wget -O data/models/mistral-8x7b-instruct.gguf URL_NOUVEAU_MODELE

# Mettre à jour .env
LLM_MODEL_PATH=data/models/mistral-8x7b-instruct.gguf

# Redémarrer
docker-compose restart api
```

4. Rebuild des Index

```bash
```

```bash
# Via API (admin only)
curl -X POST "http://localhost:8000/api/v1/admin/rebuild-index" \
  -H "Authorization: Bearer ADMIN_TOKEN"

# Ou manuellement
docker-compose exec api python scripts/rebuild_index.py
```

Sécurité

1. SSL/TLS Configuration

bash

```bash
# Générer certificat Let's Encrypt
certbot --nginx -d yourdomain.com

# Renouvellement automatique
crontab -e
0 0 1 * * certbot renew --quiet
```

2. Firewall Configuration

bash

```bash
# UFW (Ubuntu)
ufw allow 22/tcp      # SSH
ufw allow 80/tcp      # HTTP
ufw allow 443/tcp     # HTTPS
ufw enable

# Bloquer accès direct aux services internes
ufw deny from any to any port 5432  # PostgreSQL
ufw deny from any to any port 6379  # Redis
```

3. Secrets Management

bash

```bash
# Utiliser Docker Secrets (Swarm mode)
echo "your_secret" | docker secret create db_password -

# Ou AWS Secrets Manager
aws secretsmanager create-secret \
  --name rag-system/db-password \
  --secret-string "your_secret"
```

4. Rate Limiting Avancé

```python
# app/api/middleware/rate_limit.py
# Configuration par rôle
RATE_LIMITS = {
    'admin': {'calls': 1000, 'period': 60},
    'user': {'calls': 60, 'period': 60},
    'api_user': {'calls': 500, 'period': 60}
}
```

Troubleshooting

Problème 1: "Out of Memory"

Symptôme: Crash du container API **Solution**:

```bash
# Augmenter limites Docker
docker-compose.yml:
  deploy:
    resources:
      limits:
        memory: 32G

# Ou réduire batch size
CHUNK_SIZE=256  # Au lieu de 512
```

Problème 2: "Index not found"

Symptôme: Erreur lors des queries **Solution**:

```bash
# Vérifier l'existence de l'index
ls -la data/indexes/

# Rebuilder si nécessaire
docker-compose exec api python scripts/rebuild_index.py
```

Problème 3: "Slow Generation"

Symptôme: Latence > 30s **Solutions**:

1. Activer GPU:

```yaml
# docker-compose.yml
deploy:
  resources:
    reservations:
      devices:
        - driver: nvidia
          count: 1
          capabilities: [gpu]
```

2. Réduire context:

```python
LLM_CONTEXT_LENGTH=16384  # Au lieu de 32768
```

Problème 4: "Database Connection Pool Exhausted"

Symptôme: Timeout sur queries DB **Solution**:

```python
# app/core/database.py
engine = create_async_engine(
    DATABASE_URL,
    pool_size=20,        # Augmenter de 10 à 20
    max_overflow=40,     # Augmenter de 20 à 40
    pool_pre_ping=True
)
```

Performance Tuning

1. Optimisation Retrieval

```python
# Utiliser HNSW au lieu de Flat pour grandes bases
import faiss
index = faiss.IndexHNSWFlat(dimension, 32)  # M=32
index.hnsw.efSearch = 128  # Équilibre vitesse/précision
```

2. Batch Processing

```python
```

```python
# Traiter uploads par batch
@router.post("/bulk-upload")
async def bulk_upload(files: List[UploadFile]):
    # Process en parallèle
    tasks = [process_document(f) for f in files]
    results = await asyncio.gather(*tasks)
```

3. Caching Stratégique

```python
# Cache embeddings fréquents
@lru_cache(maxsize=1000)
def get_embedding(text: str):
    return embedder.embed([text])[0]
```

Checklist Pré-Production

- [ ] Variables d'environnement configurées
- [ ] Certificats SSL installés
- [ ] Backups automatiques configurés
- [ ] Monitoring actif (Prometheus + Grafana)
- [ ] Rate limiting activé
- [ ] Logs centralisés
- [ ] Tests de charge réussis (>100 req/min)
- [ ] Documentation API complète
- [ ] Plan de disaster recovery
- [ ] Alertes configurées (Slack/Email)

Support & Contact

Documentation: https://docs.votrecompagnie.com/rag-system **Email**: support@votrecompagnie.com **Slack**: #rag-system-support

---

Version: 2.0.0
Dernière mise à jour: Novembre 2025