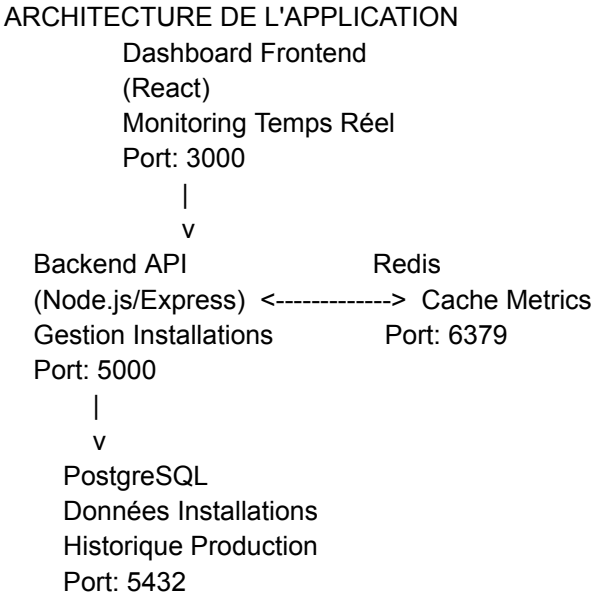
TP Docker & Kubernetes - Jour 3
Orchestration et Déploiement d'une Application Full Stack

OBJECTIFS PÉDAGOGIQUES:

À la fin de ce TP, vous serez capable de :
- Conteneuriser une application full stack multi-services
- Orchestrer plusieurs conteneurs avec Docker Compose
- Déployer une application sur Kubernetes
- Configurer le scaling et la haute disponibilité
- Mettre en place un système de monitoring basique
- Gérer les volumes et la persistance des données

CONTEXTE DU PROJET

Vous êtes développeur chez GreenWatt, une startup innovante dans le domaine des énergies renouvelables. Votre mission est de conteneurisé et déployer la plateforme de monitoring et gestion des installations solaires et éoliennes qui comprend :
- Frontend : Dashboard de monitoring React (Port 3000)
- Backend API : API Node.js/Express pour la gestion des données (Port 5000)
- Base de données : PostgreSQL pour les données des installations (Port 5432)
- Cache : Redis pour les métriques temps réel (Port 6379)



PARTIE 0 : TESTER L'APPLICATION EN LOCAL

Avant de conteneuriser l'application, vous pouvez la tester en local pour comprendre son fonctionnement.

Étape 0.1 : Installer PostgreSQL et Redis localement

Sur macOS :

```bash
brew install postgresql@15 redis
brew services start postgresql@15
brew services start redis
```

Sur Linux (Ubuntu/Debian) :

```bash
sudo apt update
sudo apt install postgresql-15 redis-server
sudo systemctl start postgresql
sudo systemctl start redis-server
```

Étape 0.2 : Initialiser la base de données

```bash
# Créer la base de données
createdb greenwatt
# Initialiser avec le script SQL
psql -d greenwatt -f database/init.sql
```

Étape 0.3 : Lancer le Backend

```bash
# Aller dans le dossier backend
cd backend
# Installer les dépendances
npm install
# Créer le fichier .env
cp env.example .env
# Modifier le .env pour utiliser localhost
# DATABASE_URL=postgresql://admin:GreenEnergy2024!@localhost:5432/greenwatt
# REDIS_URL=redis://localhost:6379
# Démarrer le serveur
npm start
```

Le backend devrait démarrer sur http://localhost:5000

Tester l'API :

```bash
# Health check
curl http://localhost:5000/api/health
# Voir les installations
curl http://localhost:5000/api/installations
# Voir les statistiques
curl http://localhost:5000/api/stats
```

Étape 0.4 : Lancer le Frontend
Dans un nouveau terminal :
```bash
# Aller dans le dossier frontend
cd frontend
# Installer les dépendances
npm install
# Créer le fichier .env
echo "REACT_APP_API_URL=http://localhost:5000" > .env
# Démarrer l'application React
npm start
```
Le frontend devrait s'ouvrir automatiquement sur http://localhost:3000
Validation : Vous devriez voir le dashboard GreenWatt avec les 10 installations d'énergies renouvelables.
Étape 0.5 : Arrêter l'application
```bash
# Arrêter le backend et le frontend (Ctrl+C dans chaque terminal)
# Arrêter PostgreSQL et Redis
brew services stop postgresql@15  # macOS
brew services stop redis
# ou
sudo systemctl stop postgresql    # Linux
sudo systemctl stop redis-server
```

PARTIE 1 : PRÉPARATION DE L'ENVIRONNEMENT

Étape 1.1 : Explorer le squelette du projet
Les fichiers de base de l'application vous sont fournis dans ce dossier. Explorez la structure :
```
TP/
├── README.md (ce fichier)
├── frontend/
│   ├── package.json
│   ├── src/
│   └── public/
├── backend/
│   ├── package.json
│   ├── server.js
│   └── routes/
└── database/
    └── init.sql
```

Étape 1.2 : Vérifier les prérequis
Vérifiez que vous avez installé :
```bash
docker --version
docker-compose --version
kubectl version --client
```

PARTIE 2 : CONTENEURISATION AVEC DOCKER

Étape 2.1 : Créer le Dockerfile pour le Backend
Objectif : Créer un Dockerfile dans le dossier backend/
Consignes :
- Utiliser l'image de base node:18-alpine
- Définir le répertoire de travail /app
- Copier package*.json et installer les dépendances
- Copier le reste du code source
- Exposer le port 5000
- Définir la commande de démarrage : npm start
Questions à vous poser :
- Pourquoi copier package.json avant le reste du code ?
- Quel est l'avantage d'utiliser l'image alpine pour une application de monitoring ?
- Comment optimiser l'image pour des déploiements fréquents ?

Étape 2.2 : Créer le Dockerfile pour le Frontend
Objectif : Créer un Dockerfile dans le dossier frontend/
Consignes :
- Utiliser une approche multi-stage build
- Stage 1 (build) :
  - Image de base : node:18-alpine
  - Installer les dépendances et builder l'application (npm run build)
- Stage 2 (production) :
  - Image de base : nginx:alpine
  - Copier les fichiers buildés depuis le stage 1 vers /usr/share/nginx/html
  - Exposer le port 80
Bonus : Créer un fichier nginx.conf personnalisé pour gérer le routing SPA

Étape 2.3 : Tester les images individuellement
Construire et tester chaque image :
```bash
# *Backend*
cd backend
docker build -t greenwatt-backend:v1 .
docker run -p 5000:5000 greenwatt-backend:v1
# *Frontend*
cd ../frontend
docker build -t greenwatt-frontend:v1 .
```

docker run -p 3000:80 greenwatt-frontend:v1
```
Validation : Vérifiez que chaque service démarre sans erreur

PARTIE 3 : ORCHESTRATION AVEC DOCKER COMPOSE (45 min)
Étape 3.1 : Créer le fichier docker-compose.yml
Objectif : À la racine du projet, créer un docker-compose.yml qui orchestre tous les services
Structure attendue :
```yaml
version: '3.8'
services:
  # Service PostgreSQL
  database:
    # À compléter

  # Service Redis
  cache:
    # À compléter

  # Service Backend
  backend:
    # À compléter

  # Service Frontend
  frontend:
    # À compléter
volumes:
  # Définir les volumes nécessaires
networks:
  # Définir le réseau
```
Consignes détaillées :

Service database :
- Image : postgres:15-alpine
- Variables d'environnement :
  - POSTGRES_DB=greenwatt
  - POSTGRES_USER=admin
  - POSTGRES_PASSWORD=GreenEnergy2025!
- Volume : Persister les données dans postgres_data
- Healthcheck : Vérifier la connexion avec pg_isready
- Réseau : greenwatt-network

Service cache :
- Image : redis:7-alpine
- Commande : redis-server --appendonly yes
- Volume : Persister les métriques dans redis_data
- Réseau : greenwatt-network
```

Service backend :
- Build : ./backend
- Dépend de : database et cache
- Variables d'environnement :
  - DATABASE_URL=postgresql://admin:GreenEnergy2024!@database:5432/greenwatt
  - REDIS_URL=redis://cache:6379
  - PORT=5000
- Ports : 5000:5000
- Réseau : greenwatt-network
- Restart : unless-stopped

Service frontend :
- Build : ./frontend
- Dépend de : backend
- Variables d'environnement :
  - REACT_APP_API_URL=http://localhost:5000
- Ports : 3000:80
- Réseau : greenwatt-network

Étape 3.2 : Lancer l'application complète

```bash
# Démarrer tous les services
	docker-compose up -d
# Vérifier les logs
	docker-compose logs -f
# Vérifier le statut
	docker-compose ps
```

Tests à effectuer :
1. Accéder au frontend : http://localhost:3000
2. Tester l'API : curl http://localhost:5000/api/health
3. Vérifier la connexion à la base de données
4. Vérifier que Redis fonctionne

Étape 3.3 : Gestion des données

Exercice :
1. Insérer des données dans la base de données via l'API
2. Arrêter les conteneurs : docker-compose down
3. Redémarrer : docker-compose up -d
4. Vérifier que les données sont toujours présentes

Question : Que se passe-t-il si vous utilisez docker-compose down -v ?

PARTIE 4 : DÉPLOIEMENT SUR KUBERNETES

Étape 4.1 : Préparer l'environnement Kubernetes
Si vous n'avez pas de cluster, utilisez Minikube :
```bash
minikube start --driver=docker
minikube status
```

Étape 4.2 : Créer les Manifests Kubernetes
Créer un dossier k8s/ avec les fichiers suivants :
1. Namespace (namespace.yaml)
Créer un namespace greenwatt
2. ConfigMap (configmap.yaml)
Stocker les configurations non sensibles :
- URL de l'API
- Paramètres de connexion (hôtes, ports)
3. Secrets (secrets.yaml)
Stocker les données sensibles (encodées en base64) :
- Mot de passe PostgreSQL
- Credentials de la base de données
Aide :
```bash
echo -n 'secret123' | base64
```
4. PersistentVolumeClaim (pvc.yaml)
Créer deux PVC :
- postgres-pvc : 1Gi
- redis-pvc : 500Mi
5. Deployments
postgres-deployment.yaml :
- Replicas : 1
- Image : postgres:15-alpine
- Variables d'environnement depuis le Secret
- Volume monté sur /var/lib/postgresql/data
- Liveness et Readiness probes
redis-deployment.yaml :
- Replicas : 1
- Image : redis:7-alpine
- Volume monté sur /data
backend-deployment.yaml :
- Replicas : 3
- Image : votre image backend
- Variables d'environnement depuis ConfigMap et Secret
- Resources limits :
  - CPU : 200m / 500m
  - Memory : 256Mi / 512Mi
- Liveness probe : /api/health
- Readiness probe : /api/ready

frontend-deployment.yaml :
- Replicas : 2
- Image : votre image frontend
- Resources limits appropriées

6. Services

postgres-service.yaml :
- Type : ClusterIP
- Port : 5432

redis-service.yaml :
- Type : ClusterIP
- Port : 6379

backend-service.yaml :
- Type : ClusterIP
- Port : 5000
- Selector : app=backend

frontend-service.yaml :
- Type : LoadBalancer (ou NodePort pour Minikube)
- Port : 80
- Target Port : 80

7. Ingress (Bonus) (ingress.yaml)

Configurer un Ingress pour exposer l'application :
- Host : greenwatt.local
- Paths :
  - / vers frontend
  - /api vers backend

Étape 4.3 : Déployer sur Kubernetes

```bash
# Appliquer tous les manifests
kubectl apply -f k8s/
# Vérifier les déploiements
kubectl get all -n greenwatt
# Vérifier les pods
kubectl get pods -n greenwatt -w
# Vérifier les logs
kubectl logs -n greenwatt deployment/backend
```

Étape 4.4 : Tester le Scaling

```bash
# Scaler le backend
kubectl scale deployment backend -n greenwatt --replicas=5
# Vérifier
kubectl get pods -n greenwatt
# Auto-scaling (bonus)
kubectl autoscale deployment backend -n greenwatt --cpu-percent=70 --min=3 --max=10
```

Étape 4.5 : Mise à jour Rolling

Simuler une mise à jour de l'application :

```bash
# Mettre à jour l'image du backend
kubectl set image deployment/backend backend=greenwatt-backend:v2 -n greenwatt
# Observer le rolling update
kubectl rollout status deployment/backend -n greenwatt
# En cas de problème, rollback
kubectl rollout undo deployment/backend -n greenwatt
```

PARTIE 5 : MONITORING ET DEBUGGING

Étape 5.1 : Commandes de debugging

Pratiquez ces commandes essentielles :

```bash
# Voir les événements
kubectl get events -n greenwatt --sort-by='.lastTimestamp'
# Décrire un pod
kubectl describe pod <pod-name> -n greenwatt
# Accéder à un conteneur
kubectl exec -it <pod-name> -n greenwatt -- /bin/sh
# Voir les logs en temps réel
kubectl logs -f <pod-name> -n greenwatt
# Port-forward pour tester
kubectl port-forward service/backend 5000:5000 -n greenwatt
```

Étape 5.2 : Résolution de problèmes

Scénarios à tester :

1. Pod en CrashLoopBackOff : Que faire ?
2. Service inaccessible : Comment débugger ?
3. Base de données ne démarre pas : Vérifier les volumes
4. Application lente : Analyser les ressources

PARTIE 6 : BONUS ET AMÉLIORATIONS

Bonus 1 : CI/CD avec GitHub Actions

Créer un workflow qui :

- Build les images Docker
- Push vers Docker Hub
- Déploie automatiquement sur K8s

Bonus 2 : Monitoring avec Prometheus

- Installer Prometheus sur le cluster
- Exposer des métriques depuis le backend
- Créer des dashboards Grafana

Bonus 3 : Sécurité

- Implémenter Network Policies
- Scanner les images avec Trivy
- Utiliser des non-root users dans les conteneurs

Bonus 4 : Helm Charts

- Créer un Helm Chart pour l'application
- Gérer différents environnements (dev, staging, prod)

LIVRABLES ATTENDUS

À la fin du TP, vous devez avoir :

1. Dockerfiles fonctionnels pour frontend et backend
2. docker-compose.yml orchestrant tous les services
3. Manifests Kubernetes complets dans le dossier k8s/
4. Application déployée et accessible sur Kubernetes
5. Documentation : Un fichier RAPPORT.md expliquant :
   - Les choix techniques effectués
   - Les difficultés rencontrées et solutions
   - Les commandes utilisées
   - Screenshots de l'application fonctionnelle

RESSOURCES UTILES

- Docker Documentation : https://docs.docker.com/
- Docker Compose Reference : https://docs.docker.com/compose/compose-file/
- Kubernetes Documentation : https://kubernetes.io/docs/
- Kubectl Cheat Sheet : https://kubernetes.io/docs/reference/kubectl/cheatsheet/

AIDE ET SUPPORT

En cas de blocage :

1. Vérifiez les logs : docker logs ou kubectl logs
2. Consultez la documentation officielle
3. Utilisez kubectl describe pour voir les événements
4. Demandez de l'aide à votre formateur

Bon courage !