Rapport Technique de Déploiement du projet AutoScaling et IaC

Infrastructure Kubernetes Complète

Réalisé par Maria Saad et Sinem Sevinc

4 avril 2025

---

1. Objectif du Projet

Ce projet vise à concevoir et déployer une infrastructure cloud-native moderne reposant sur des microservices conteneurisés. Chaque composant est isolé, scalable et monitoré via un cluster Kubernetes. L'ensemble est conçu pour être reproductible, automatisé, et conforme aux meilleures pratiques DevOps.

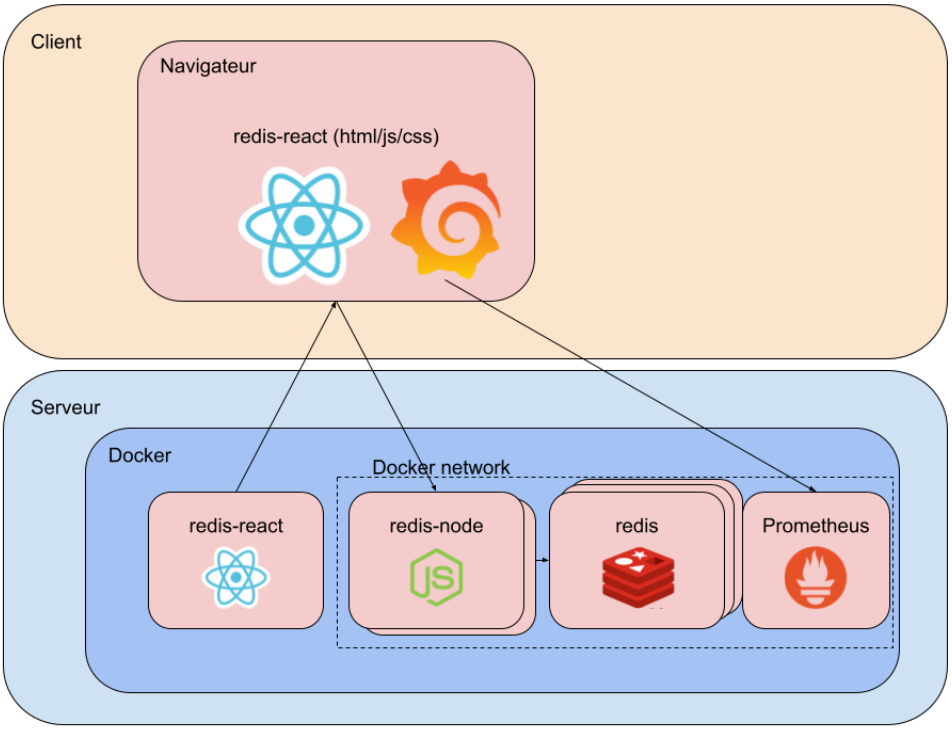
Technologies clés : Kubernetes, Docker, Redis, Node.js, React, Prometheus, Grafana, HPA, VPA.

2. Contenu du Dépôt Git

- Fichiers `.yaml` pour Kubernetes : déploiements, services, autoscalers
- Dockerfiles pour les services Node.js et React
- Scripts d'automatisation : `script.sh`, `script_react.sh`
- Fichier `README.md` expliquant la procédure
- Rapport PDF technique
- Images Docker hébergées sur Docker Hub

3. Architecture de l'Infrastructure

L'architecture globale du projet est illustrée ci-dessous. Elle montre les différents composants (Redis, Node.js, React, Prometheus, Grafana) et leurs interactions via les services Kubernetes.



4. Déploiement de l'Infrastructure

4.1. Fichiers YAML et leur rôle

Le déploiement de l'infrastructure repose sur plusieurs fichiers YAML, chacun d'écrivant un composant spécifique. Voici leur rôle :

Redis

- **redis-master-deployment.yaml** : Création du pod Redis Master
- **redis-slave-deployment.yaml** : Création de deux réplicas de Redis Slave avec synchronisation sur le master.
- **redis-services.yaml** : Exposition des services Redis Master et Slave via un service Kubernetes.

Node.js Backend

- **nodejs-deployment.yaml** : Déploiement de deux pods Node.js avec connexion á Redis Master et Slave.
- **nodejs-service.yaml** : Service LoadBalancer exposant l'API sur le port 8080.

Frontend React

- **react-deployment.yaml** : Déploiement de l'application React avec 2 réplicas.

— **react-service.yaml** : Service LoadBalancer exposant React.

Monitoring avec Prometheus et Grafana

— **grafana-deployment.yaml** : Déploiement de Grafana avec stockage temporaire.
— **grafana-service.yaml** : Service LoadBalancer exposant Grafana sur le port 3000.
— **prometheus-configmap.yaml** : Configuration de Prometheus pour collecter les métriques des services.
— **prometheus-deployment.yaml** : Déploiement de Prometheus avec volume monté.
— **prometheus-service.yaml** : Service LoadBalancer exposant Prometheus sur le port 9090.

4.2. Initialisation du Cluster Kubernetes

1. Démarrer Minikube :
```
minikube start
```
2. Accéder au dossier `Kubernetes` contenant les fichiers :
```
cd Kubernetes
```
3. Rendre le script exécutable :
```
chmod +x script.sh
```
4. Lancer le déploiement avec :
```
./script.sh
```

Ce que fait le script `script.sh` :

Le script automatise l'installation de plusieurs composants essentiels :
— Déploiement de Redis (master, slave, service et exporter de métriques)
— Déploiement de l'API backend Node.js
— Déploiement de Prometheus et Grafana pour le monitoring
— Lancement de `minikube tunnel` pour exposer les services de type LoadBalancer
— Récupération de l'IP externe du service Node.js
— Affichage de la liste des pods et services actifs
Une fois ces actions complétées, les adresses IP externes sont prêtes à être utilisées pour configurer le frontend et accéder aux services de monitoring.

5. Configuration du Frontend React

Avant de construire l'image Docker de l'application React, il est essentiel de configurer correctement l'URL du backend pour éviter d'intégrer une mauvaise IP dans l'image.
Pourquoi ? Car cette variable est codée dans le bundle React au moment du build, elle ne peut donc pas être modifiée dynamiquement après déploiement.

1. Récupérer l'adresse IP externe du backend Node.js (affichée par le script ou avec `kubectl get svc`)

2. Aller dans `react/redis-react/src`
3. Modifier le fichier `conf.js` :

```
export const URL = 'http://<NODEJS_IP>:8080';
```

4. Remplacer `<NODEJS_IP>` par l'adresse IP externe obtenue

6. Création et Publication de l'image Docker React

```
cd react/redis-react
docker build -t react-app:2.0 .
docker login
docker tag react-app:2.0 <DOCKER_USERNAME>/react-app:2.0
docker push <DOCKER_USERNAME>/react-app:2.0
```

6.1. Mise à jour de l'image dans le fichier YAML

Dans `react-deployment.yaml`, modifier :

```
image: <DOCKER_USERNAME>/react-app:2.0
```

6.2. Déploiement de l'application React

```
cd ../../Kubernetes
chmod +x script_react.sh
./script_react.sh
```

7. Ajout de Prometheus dans Grafana

1. Accéder à Grafana : `http://127.0.0.1:3000` (admin/admin)
2. Aller dans Configuration > Data Sources
3. Cliquer sur `Add data source` et sélectionner Prometheus
4. Entrer l'URL : `http://prometheus.default.svc.cluster.local:9090`
5. Cliquer sur `Save & Test` pour valider la connexion

8. Montée à l'échelle dynamique

— **HPA** : ajuste dynamiquement le nombre de pods selon la charge CPU
— **VPA** : ajuste automatiquement les ressources CPU/mémoire des pods

9. Accès aux Services

— **API Node.js** : http://127.0.0.1:8080
— **Frontend React** : `http://127.0.0.1:<PORT>` (via `kubectl get svc`)

— **Prometheus** : http://127.0.0.1:9090
— **Grafana** : http://127.0.0.1:3000 (admin/admin)

10. Conclusion

Ce projet met en œuvre une infrastructure DevOps complète, modulaire et scalable. Tous les composants sont conteneurisés et intégrés dans Kubernetes, avec un monitoring complet et une adaptabilité automatique à la charge grâce aux autoscalers.

Tous les fichiers sont disponibles dans le dépôt Git du projet.